Основы разработки ПО

Введение в процесс создания ПО

Кулаков Кирилл Александрович

Информация о курсе

- Фокус дисциплины:
  - Методология проектирования программного обеспечения
  - Методология тестирования программного обеспечения
- Лекции http://cs.petrsu.ru/~kulakov/courses/develop
- Лабораторные работы:
  - Изучение инструментов автоматизации тестирования
  - Разработка и тестирование приложения
- Практика по дисциплине:
  - Инструментарий & моделирование & реализация
- Помощь
  - Сайт курса (http://cs.petrsu.ru/~kulakov/courses/develop)
  - График консультаций кафедры ИМО (215 ауд.)
  - Электронная почта (kulakov@cs.petrsu.ru)

Определения

- **Программное обеспечение** (ПО) - множество развивающихся во времени логических предписаний, с помощью которых некоторый коллектив людей управляет и использует многопроцессорную и распределенную систему вычислительных устройств.
  - Логические предписания – это не только сами программы, но и различная документация и т. д. (система отношений)
  - Современное ПО предназначено, как правило, для одновременной работы со многими пользователями, которые могут быть значительно удалены друг от друга в физическом пространстве (распределенность)
  - Задачи решаемые современным ПО, часто требуют различных вычислительных ресурсов (ресурсоемкость)
  - ПО развивается во времени – исправляются ошибки, добавляются новые функции, выпускаются новые версии, меняется его аппаратная база (обновляемость)

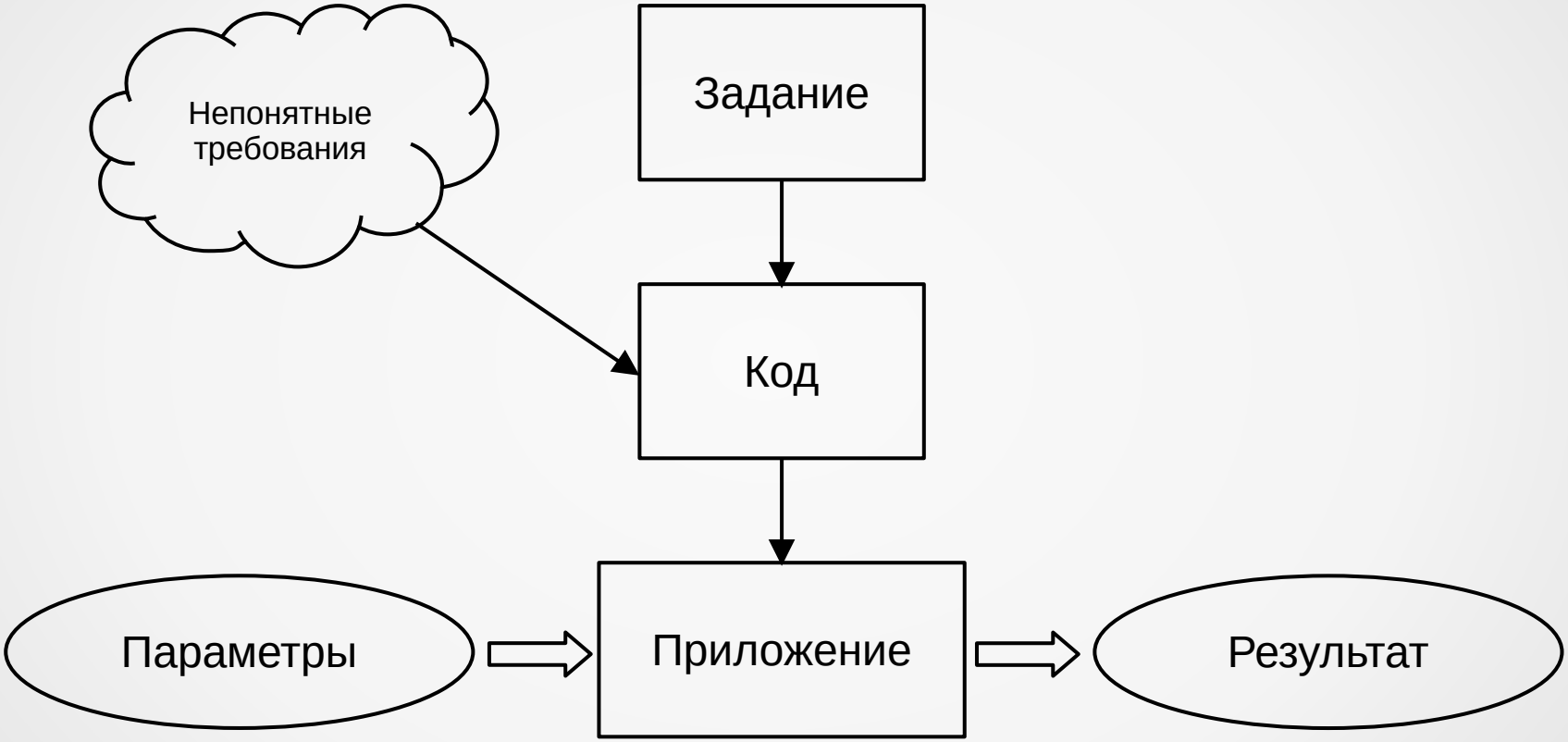
Виды программ

- Автономное (standalone) консольное ПО
- Автономное графическое ПО
- Серверное ПО
  - Инфраструктурное ПО
  - Клиентское ПО
- Сервисы/демоны/процессы
- Мобильное ПО
- Встраиваемое (embedded) ПО

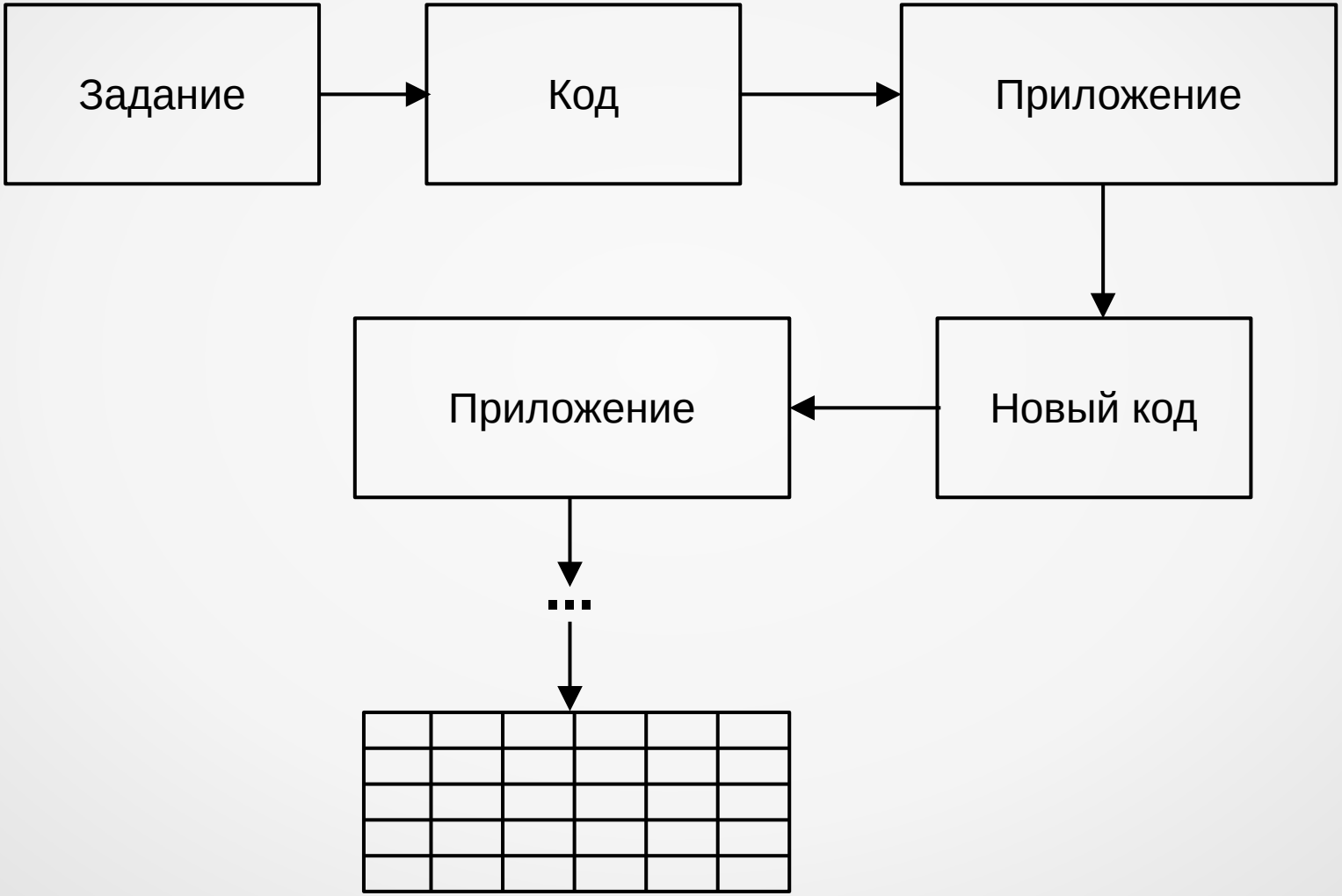
Свойства ПО

- **Сложность** программных объектов, которая существенно зависит от их размеров.
- **Согласованность** – ПО основывается не на объективных посылках, а должно быть согласовано с большим количеством интерфейсов, с которыми впоследствии оно должно взаимодействовать.
- **Изменяемость** – ПО легко изменить и, как следствие, требования к нему постоянно меняются в процессе разработки.
- **Незримость** – ПО невозможно увидеть, оно виртуально. Поэтому, например, трудно воспользоваться технологиями, основанными на предварительном создании чертежей, успешно используемыми в других промышленных областях

Процесс создания простой программы



Более сложный вариант



Программа с развитием

Проблемы написания сложных программ

- Техническое задание
  - Слишком сложное
  - Очень большое
- Сроки реализации
- Потребление ресурсов (время, память, сеть, экран, …)
- Объем работ
  - 1 программист в среднем пишет 500 строк хорошего кода в день
- Взаимодействие внутри команды

- Необходимость перехода от программирования как искусства к программированию как индустрии в связи с усложнением программного обеспечения.

Проблемы написания сложных программ

- Цели программ бывают разные
  - На практике 1 программа = множество целей
- Задача разработчика ПО — приблизиться к цели
  - Реализация качественного ПО
- Как проверить результат работ?
  - Выполнение тестирования и аттестации

- Необходимо предусмотреть возможность проверок кода

Обеспечение взаимодействия

- Сложные программы создаются командами разработчиков
- Необходима организация взаимодействия людей
  - На уровне задач (кто что будет делать)
  - На уровне работ (кто как делает)
  - На уровне результатов (как оно должно быть)

- Необходима организация взаимодействия программ
  - Объекты данных
  - Процессы
  - Сценарии

Обеспечение взаимодействия

- **Законодательство** (международное, РФ)
  - Пример: ФЗ №63-ФЗ от 06.04.2011 (ред. От 30.12.2015) «Об электронной подписи»
- **Стандарты** (международные, отраслевые)
  - ЕСПД (ГОСТ 19) Единая система программной документации
  - ГОСТ 34. Автоматизированные системы
  - Стандарты IEEE, например, 829-1998 «Стандарт для тестовой документации тестирования программного обеспечения»

Обеспечение взаимодействия

- **Руководства** (Guidelines):
  - Рекомендации по кодированию
  - Рекомендации по интерфейсу пользователя
  - Рекомендации по проектированию
  - Рекомендации по тестированию
  - …
- Ожидания пользователя
  - «Если кнопочка нарисована она должна нажиматься.»

Обеспечение взаимодействия

- Взаимодействие между программами
  - **Программные интерфейсы** (API): соглашение между разработчиками о подключении программ друг к другу.
    - Пример: математическая библиотека (#include <math.h>, ...)
  - **Протоколы передачи данных**: описание процессов передачи и структур данных.
    - Пример: HyperText Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), Remote Desktop Protocol (RDP), Remote Procedure Call (RPC)

Использование готовых решений

- Современная разработка ПО не начинается с нуля
- Если взять два соседних проекта, то у них будет что-то общее
  - Пример: чтение файла специфичной структуры

- Объединение повторяющихся кусков кода в **функции**
- Объединение функций в **библиотеки** (модули)
- Использование «решений» прошлых разработок
  - **Шаблоны** проектных решений
  - **Шаблоны** кода

Использование готовых решений

- Идеальный результат: берем блоки готовых решений, замазываем их клеем из кода и получаем построенную программу

- **Программные каркасы** (фреймворки): скелет будущего приложения куда надо вставить кирпичики кода
  - Уже готовые решения по стандартным операциям
  - Соответствие стандартам, рекомендациям и т. п.
  - Наличие документации

Использование готовых решений

- Программные каркасы позволяют:
  - Минимизировать объем работ
  - Получить работающее приложение «за 5 минут»
  - Повторно использовать свои наработки
  - Расширять функционал с помощью дополнительных библиотек
  - Соответствовать стандартам и рекомендациям

- **Программные каркасы не решат за вас задачу, они помогут быстрее сделать решение!**

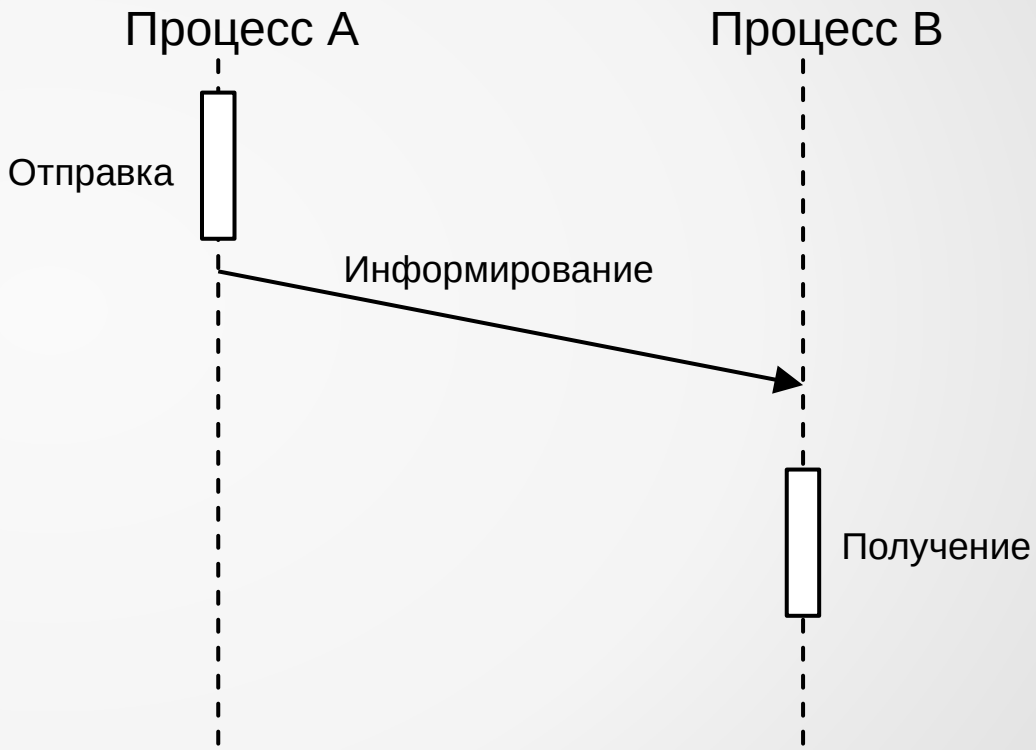
Процесс во времени

- Пример часто встречающейся проблемы: работа приложения во времени
  - Выполнение длительных алгоритмов
  - Прерывания во время выполнения
- Результат: нелинейная работа кода

- **Синхронное программирование** — последовательное выполнение операций с ожиданием результата (блокировка)
- **Асинхронное программирование** — результат работы операции доступен не сразу же, а через некоторое время

Процесс во времени

- Пример: передача массива данных между процессами
  - Процесс А отправляет массив
  - Процесс В получает массив
- Как отправить массив без больших затрат ресурсов?
- Как узнать что массив отправлен?
- Как получить массив?



Процесс во времени

- Код процесса A:

```
sendData(arrayData).onSuccess(
    function () {notifyProcessB();}
)
```

- Код процесса B

```
waitData().onReady(
    function (array arrayData) {m_data = arrayData;}
)
```

Процесс во времени

- Разработчик/программист должен всегда представлять в какой момент времени какой кусок кода будет выполняться и какие данные ему будут доступны
- Иначе он столкнется с различными проблемами, например:
  - Проблема узкого места bottleneck
  - Проблемы взаимных блокировок
- Решение: продумать (спроектировать) приложение до его реализации

Программная инженерия

- **Программная инженерия** — раздел компьютерных наук (англ. computer sciences), изучающий методы и средства построения компьютерных программ как продукта теоретической и инженерной деятельности разработчиков или их коллективов.
- **Основные критерии ПИ:** продуктивность, индустрия и качество.

Программная инженерия

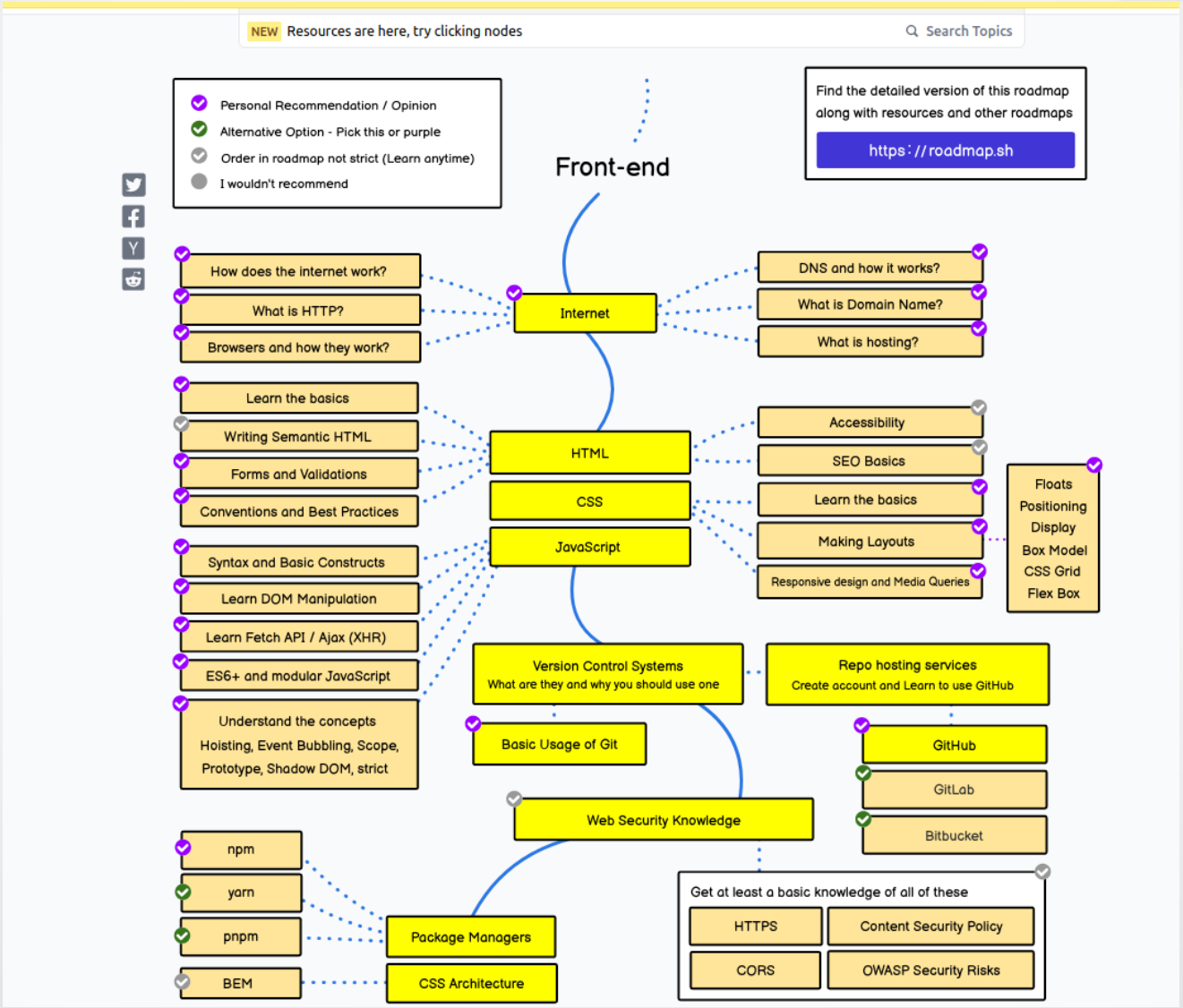
- Программная инженерия основывается на математических дисциплинах:
  - теория алгоритмов нормальные алгоритмы, вычислимые функции, машина Тьюринга, граф-схемы, модели алгоритмов;
  - математическая логика формальный вывод утверждений;
  - теория управления принципы, методы и общие законы планирования и управления в сложных системах;
  - теория доказательств математическая теория вывода по аксиомам и утверждениям, теория верификации программ;
  - теория множеств формальное представление совокупностей объектов из предметной области.

Выводы

- Для того чтобы разрабатывать качественные приложения необходимо:
  - Знать стандарты, рекомендации, правила
  - Уметь оптимизировать затраты на программирование с использованием функций, библиотек, фреймворков
  - Уметь выполнять проектирование до реализации
  - Уметь проверять полученные решения

Выводы

- https://roadmap.sh/frontend



Выводы

- https://roadmap.sh/backend