# Примерная программа курса

## Личное мастерство

1. [-] Вводная лекция + Рабочее место программиста
   * 4 кита: браузер, файл-менеджер, текстовый редактор, консоль, [IDE]
   * Браузер: облачные сервисы (почта, документы), StackOverflow, Google
   * Файл-менеджер: FTP, копирование путей, быстрая навигация
   * Cинхронизаторы директорий (Google Drive, BitTorrent Sync)
   * Искалки и запускалки
   * Other: файловые шары, виртуальные машины
2. [+] Текстовые форматы [Кирилл Корняков]
   * Текстовый и бинарный формат, сравнение
   * Популярные форматы на основе текстового: txt, xml, yaml, json
   * Текст как исходник: scripts, GraphVis, PlantUML, latex, doxygen
   * Легковесные языки разметки: Markdown, Textile, MediaWiki, ReST
   * Markdown: Wiki, книги, Slides, Web-publishing (Jekyll)
3. [+] Обработка текста / Текстовые редакторы [Кирилл Корняков]
   * Редакторы: vim, Emacs, Sublime
   * Манипуляции (навигация, сортировка, сниппеты и макросы)
   * Regular expressions
   * Утилиты командной строки (find, grep, sed, awk, head/tail/less)
4. [+] Автоматизация: командная строка и скриптовые языки [Сергей Носов]
   * UNIX philosophy
   * Bash (cron, watch), other shells, настройка под себя, Tilda
   * Python (xml, excel, pdf, matplotlib)
   * Анонс: make, unit-testing, CI
5. [+] Системы контроля версий [Кирилл Корняков]
   * Введение и история
   * Subversion, Mercurial
   * Git как DVCS и не только
   * Примеры хуков
   * Тренинг (книга Pro Git)

## Работа с кодом

1. [-] Написание кода
   * IDE с последними функциями (подсветка, автозавершение)
   * Рефакторинг
   * Автогенерация кода, метапрограммирование
2. [+] Создание "проектов" [Александр Шишков]
   * IDE
   * Билд-системы
     + IDE
     + Makefiles
     + CMake
     + Other: Rake, scons, waf
3. [.] Построение исполняемых модулей [Александр Сморкалов]
   * Обзор: компиляторы + линкеры, трансляторы, виртуальные машины
   * GNU toolchain, понятие кросс-компиляции
   * Компиляторы
     + MSVS
     + GCC, clang, LVVM
4. Анализ бинарных модулей [Александр Сморкалов]
   * Статическая и динамическая линковка
   * Зависимости
   * API, binary compatibility
   * Дизассемблирование

## Качество кода

1. [?] Качество кода
   * IDE
   * Системы для документирования кода (Sphinx, Doxygen)
   * Статический и динамический анализ кода
   * Переносимость
2. [?] Отладка [Никита Манович]
   * Логирование (printf)
   * Отладчики
3. [+] Тестирование [Кирилл Корняков]
   * Системы для написания Unit-тестов
     + JUnit (XML format)
     + GoogleTest
4. [+] Непрерывная интеграция [Кирилл Корняков]
   * BuildBot
   * Jenkins
   * GitHub (TravisCI)
5. [?] Профилирование и оптимизация производительности [Иосиф Мееров]
   * Замеры времени
   * Техники оптимизации
     + устранение проблемы или вызов библиотек
     + алгоритмическая
     + общие техники
     + векторизация
     + многопоточность (по данным, по задачам)
     + многопроцессность

## Коллективная разработка

1. [.] Командная разработка [Кирилл Корняков]
   * Примеры Git workflow
   * Трекеры задач (Redmine, Trac, Bugzilla)
   * Инструменты для peer review (Gerrit)
   * GitHub (GitLab), Bitbucket, other (Code Google, SourceForge)
2. [.] Формирование сообщества [Александр Шишков]
   * Листы рассылки
   * Форумы, Q&A
   * UserEcho
   * Социальные сети

# Практика

Разработка кросс-платформенной библиотеки на С++, с Sphinx документацией и CTest и GoogleTest тестами, построением при помощи make и CMake, и автоматическими прогонами тестов при помощи TravisCI.

Каждая новое задание дается через 2 недели. Итого, через 10-12 недель все должно быть сдано.

1. Клонировать проект на GitHub, создать README в формате Markdown. Там должны присутствовать следующие секции:
   * Тема лабораторной работы
   * Информация об авторе
   * Объяснение раскладки по подпапкам в виде списка
   * Перечисление возможностей проекта
2. Проектирование
   * Разработать интерфейс класса (hpp-файл)
   * Задокументировать его в стиле Sphinx, см. заготовку: [исходник](rst_example), [результат](rst_out)
   * Можно:
   * PlantUML попробовать прикрутить
   * Еще и Doxygen поднять
3. Реализация
   * Реализовать класс, поместить его в библиотеку (статическую?).
   * Добавить тестирующее консольное приложение, которое проверяло бы класс на нескольких наборах данных, в том числе некорректных.
   * Организовать построение при помощи make.
   * Убедиться, что присланный пулл-реквест проходит построение на TravisCI.
4. Интеграция
   * Добавить построение при помощи CMake
   * Добавить несколько тестов на CTest
   * Обеспечить запуск в рамках инфраструктуры TravisCI
   * Убедиться что пулл-реквест зеленый
5. Тестирование (возможно стоит поменять с предыдущим пунктом)
   * Добавить unit-тесты с использованием фреймворка GoogleTest, обеспечить их запуск и выполнение в рамках общего тестирования TravisCI
   * Выделить класс Application, также покрыть его модульными тестами
6. Портирование
   * Создать Python-обертки для своего класса (и возможно класса-приложения)
   * Протестировать его при помощи модульных тестов Python

Возможно:

1. Добавить генерацию Java и Python интерфейсов
2. Можно парсить XML-вывод от GoogleTest (будет один скрипт на всех)
3. Рекомендовать использовать Qt Creator (кстати как альтернатива Cygwin)