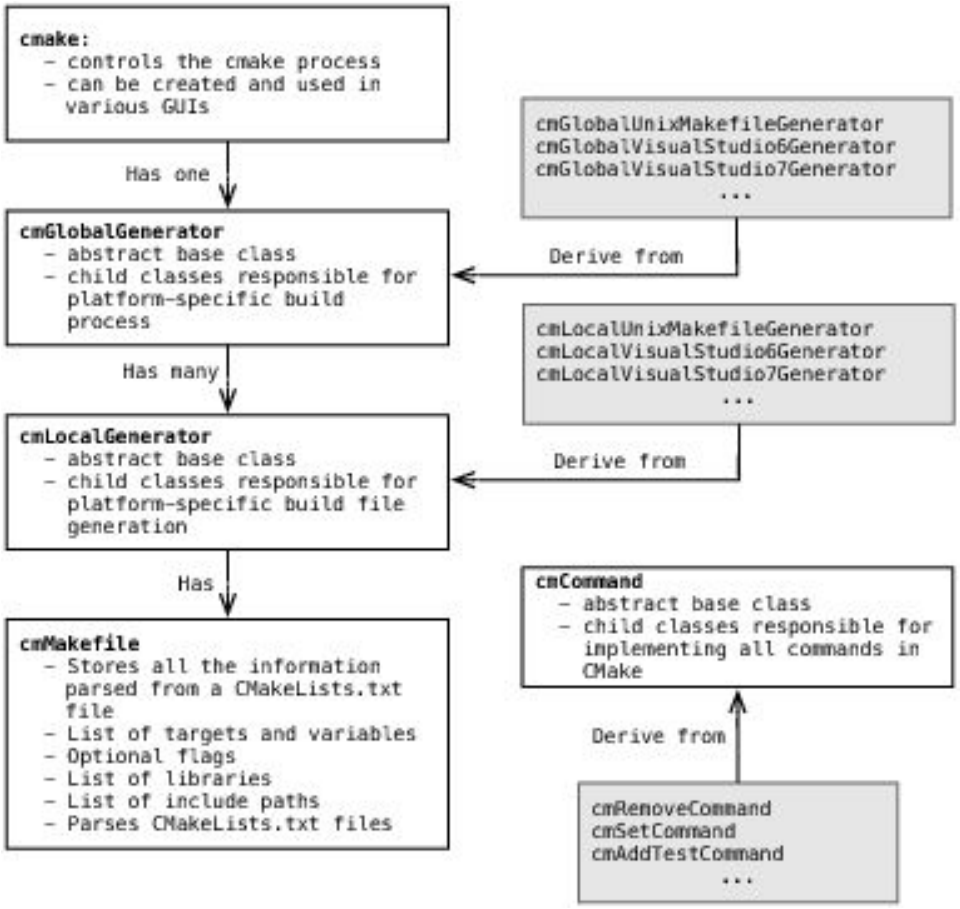
http://aosabook.org/en/cmake.html
http://aosabook.org/en/selenium.html
http://aosabook.org/en/mercurial.html
http://aosabook.org/en/integration.html

CMake:

1. Контекст:
    a. Проекты на C++, хотелось управляемый процесс сборки на разных платформах
    b. Проблема с Makefile: много писать, сложно разбираться
2. Требования:
    a. Зависимость от компилятора -> написали язык для конфигурационных файлов
    b. Генерация проектных файлов для Visual Studio
    c. Сборка для разных целей (с помощью ключей)
    d. Составные этапы сборки
    e. Различные варианты дерева сборки (debug-сборку хотим в отдельной папке)
    f. автоматическое сканирование зависимостей заголовочных файлов
    g. кроссплатформенность
3. Основные архитектурные решения (процесс в два этапа):
    a. Configure
        i. Промежуточное представление процесса сборки
        ii. CMakeCache.txt -- захешированное состояние предыдущих сборок (пустое, если в первый раз)
        iii. CMakeLists.txt -- перечисляются те файлы, которые будут участвовать в процессе сборки (вычитывает все флаги, понимает что делать, записывает это)
        iv. паттерн “Команда”
    b. Generate -- построение всех артефактов сборки:
        i. создание файлов для сборки под целевое окружение



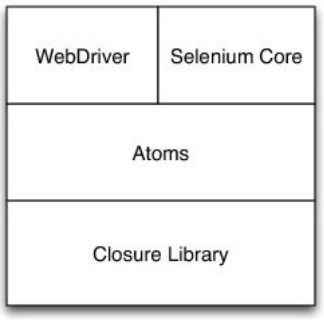
c.

4. Особенности реализации:
    a. Не было времени на продумывание архитектуры
    b. Нужна обратная совместимость
    c. Собственный скриптовый язык
    d. Плагины на плюсах не зашли
    e. Слишком широкий API -- все нужно поддерживать

Selenium -- средство для автоматизации GUI-тестов для веб-приложений:

1. Контекст:
    a. Основные инструменты:
        i. WebDriver -- программный интерфейс для взаимодействия с браузером
        ii. Selenium IDE -- позволяет записать сценарий на сайте и проигрывать потом
        iii. Selenium Grid -- позволяет строить инфраструктуру серверов для тестирования
    b. Код выполняется в песочнице, т.к. нельзя ему давать много прав, но хочется:
        i. Selenium Remote Control -- манипулируем HTML на JS
        ii. WebDriver -- взаимодействие со стороны через паттерн Адаптер
2. Требования:
    a. Эмуляция действий пользователя
    b. Поддержка разных браузеров

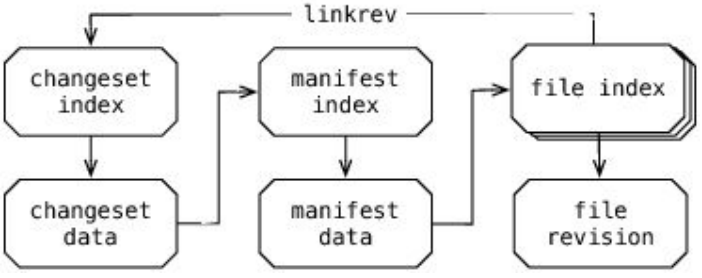
   c. Bus factor -- нужно повышать (сколько человек могут уйти, а система продолжит работать) -> хорошая документация, понятная архитектура
3. Основные архитектурные решения:
   a. Role-based интерфейс
   b. Многоуровневая архитектура разграничивает абстракции
   c. Closure Library -- примитивная работа с JS
   d. Atoms -- уровень моделей элементов (манипуляция свойствами, события)
   e. Если нельзя напрямую работать с браузером -- Selenium RC (чистый JS, сервер как прокси к браузеру, асинхронный Responce/Request)



4. Особенности реализации:
   a. Большая работа по унификации браузеров
   b. Уровни абстракции для унификации
   c. Спрятали всю сложность в драйвер, но их сложно реализовывать

Mercurial:

1. Контекст:
   a. распределенная VCS
   b. централизованность плоха тем, что при смерти сервера нет доступа к проекту
2. Требования:
   a. миллионы файлов/ревизий/разработчиков/времени использования
   b. компрессия хранилища данных
   c. работа с произвольными ревизиями (поиск по ним)
   d. добавление новых ревизий, откат к старым
3. Основные архитектурные решения:
   a. Revlog: файл индекса ревизий, файл данных
   b. Модель данных (добавление снизу-вверх):
      i. changelog -- метаданные о ревизии
      ii. manifests -- имя файла в ревизии + ссылка на filelog
      iii. filelog -- содержимое файлов в ревизии
      iv. dirstate -- текущая ревизия
      v.



   c. Версионирование:
      i. Ветка путем клонирования репозитория
      ii. Bookmarks
      iii. Именованные и анонимные ветки

    d. Архитектура -- консольная утилита: словарь принимает команды, диспетчер про них знает и вызывает, localrepo производит манипуляции с репозиторием
4. Особенности реализации:
    a. Меньше команд, чем в гите
    b. Другая модель данных, разные виды сжатия