Современные технологии разработки ПО

Введение

Лектор: Петров Антон Александрович
petrov.a@kubsau.ru

Состав курса

1. Разработка сложного ПО с использованием техники предметно-ориентированного проектирования
2. Основы современных технологий разработки ПО, DevOps
3. **Зачёт**

Литература (общая)

1. Иванова Г.С. Технология программирования: Учебник. - М: Кнорус, 2014. – 333 с.
2. Иванова Г.С., Ничушкина Т.Н. Объектно-ориентированное программирование: Учебник для вузов. - М.: МГТУ, 2014. – 368 с.
3. Фаулер М. UML. Основы, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 192 с.
4. Бек Кент, Фаулер Мартин, Брант Джон. Рефакторинг. Улучшение проекта существующего кода. – Вильямс. 2017
5. Стив Макконел. Совершенный код. Мастер-класс. — М. : Издательство «Русская редакция», 2010. — 896с.
6. C++ reference. - URL: https://en.cppreference.com
7. Портал C++. - URL: http://www.cplusplus.com
8. Методичка по данному курсу

Литература (модуль 1)

1. Meyer, Bertrand. Object-Oriented Software Construction, second edition. - Prentice Hall, 1997
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2015. — 368 с.
3. Эванс Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. – СПб.: ООО «Диалектика», 2019. – 448 с.

Основные инструменты

- C++11 (желательно C++17)
- UML 2.0 (классы, последовательности, пакеты)
- Любая ОС и среда разработки (Linux предпочтительнее)
- git/GitLab: https://bmstu.codes
- Рекомендации:
  - GNU/Linux Ubuntu 18.04 LTS или 19.04
  - Qt Creator или Visual Studio Code
  - GCC (g++)

Расширения Visual Studio Code

- C/C++ for Visual Studio Code
  - Microsoft. *This preview release of the C/C++ extension adds language support for C/C++ to Visual Studio Code, including features such as IntelliSense and debugging*
- C/C++ Compile Run extension
  - Danielpinto8zz6. *An extension running on Visual Studio Code to Compile & Run single c/c++ files easly*
- Spelling Checker for Visual Studio Code (Russian)
  - Street Side Software. *A basic spell checker that works well with camelCase code*

Суть курса

- Дать основные приёмы при создании:
  - сложного ПО,
  - качественного ПО,
  - в условиях гибкой разработки.
- Введение в современные технологии (в том числе DevOps) с акцентом на тематику нашей кафедры
- Дать основы безопасного кодирования на C++ и подобных языках

Что будет в курсе

- Основы ООП
- Основы современного C++
- Система управления версиями Git и GitLab
- Предметно-ориентированное проектирование (Domain Driven Design, DDD)
- Введение в архитектуру распределённых приложений
- Микросервисы
- DevOps
- Сравнение класических и гибких методологий

Основные признаки сложного ПО *

- **Комплексность**,
  - то есть, собственно, сложность, которая проявляется в большом количестве объектов, их вложенности друг в друга и связями между ними
- **Длительный жизненный цикл**,
  - то есть необходимость не только разработать ПО, но и поддерживать его развитие на протяжении длительного времени
- **Работа в команде**,
  - что подразумевает невозможность разработать и поддерживать сложное ПО в одиночку за приемлемое время, а также устойчивость проекта в условиях текучки команды разработки

How the UML diagram describes the software

How the code is actually written

Примеры сложных проектов

Основы гибкой разработки

- Итеративная модель жизненного цикла:
  - классические методологии,
  - гибкие методологии.
- Agile — это ценности и принципы (не методология)
  - манифест гибкой разработки: http://agilemanifesto.org
- Методологии гибкой разработки:
  - XP (экстремальное программирование),
  - Scrum,
  - Agile Unified Process (AUP),
  - и многие, многие другие.

Ценности Agile *

1. Люди и взаимодействие важнее процессов и инструментов
2. Работающий продукт важнее исчерпывающей документации
3. Сотрудничество с заказчиком важнее согласования условий контракта
4. Готовность к изменениям важнее следования первоначальному плану

Резюме по гибкой разработке

- Постоянное уточнение требований, а значит изменение кода — приветствуются
- Минимизация проектной и конструкторской документации — приветствуется
- «Грязное» проектирование (см. Стив Макконел, Совершенный код) — в порядке вещей
- Итеративный подход, постоянное усложнение и принцип Ready to show с самого начала проекта — обязательно!
- →
- **Код должен быть изначально приспособлен к изменениям!**

Адаптивный код

- Понятен:
  - д.б. соглашения о стилях кодирования,
  - самодокументируемый код.
- Легко изменяем:
  - минимизация связей,
  - упрощение классов,
  - минимизация состояний,
  - и др.
- Устойчив к изменениям:
  - изменения в одной функциональности не должны ломать другую функциональность

Почему C++

- C++ позволяет писать адаптивный код
- У вас был курс C++
- Высокоуровневый ЯП с хорошей поддержкой ООП
- Высокоэффективный высокоуровневый ЯП
- Широко распространён
- Развивается, особенно, начиная со стандарта C++11
- Наиболее открытый ВЯП, т. е. содержит минимальное количество встроенных конструкций
- C++ позволит заглянуть в детали реализации некоторых конструкций и понять различия их реализации в других ЯП

Заблуждения о C++

- №1: «C++ нишевый язык, на нём больше не пишут программ», однако:
  - поисковые движки, высоконагруженные приложения;
  - игры и игровые движки (иногда, наружу C#);
  - браузеры;
  - спецэффекты и анимация;
  - компиляторы (не только C++), виртуальные машины;
  - научные программы (CERN и бозон Хикса);
  - части ОС (драйверы, userspace);
  - автопром, авиапром, медицина, заводы, биржа;
  - графические редакторы, 3D-сканеры, сапромат;
  - офисные приложения;
  - и так далее!

Светлая и тёмная стороны C++

- Основной постулат философии C++: «Если разработчик хочет выстрелить себе в ногу, ЯП не будет ему мешать»
- Отсюда вытекают две стороны C++:
  - светлая и тёмная,
  - доступная и скрытая для большинства,
  - доктор Джекил и мистер Хайд C++,
  - Гарри Поттер и Воландеморт C++.
- Наш путь — по светлой стороне C++, но
  - иногда, балансируя на границе между светом и тьмой, мы будем осторожно изучать тёмную сторону C++, рассматривать компромиссы и оценивать потерю эффективности, в том числе, в сравнении с другими ЯП

ООП

- Доминирующая парадигма при разработке сложного ПО
- Позволяет проектировать адаптивный код
- Имеет широкое распространение
- Хорошо описана во многих источниках
- Вы её проходили

Основные принципы ООП*

- Абстрагирование
- Инкапсуляция
- Наследование
- Полимирфизм

Основные определения *

- «Объект» = «Экземпляр класса»
- **Инкапсуляция** — упаковка данных и методов в единый компонент
- «Инкапсуляция» ≠ «сокрытие»
- **Сокрытие** — принцип проектирования, заключающийся в разграничении доступа различных частей программы к внутренним компонентам друг друга

Состояние объекта *

- **Состояние объекта** — совокупность значений (состояний) членов класса и состояний базовых классов
- **Изменение состояния объектов** — изменение значения (состояния) любого члена класса или состояния базового класса
- **Некорректное / недопустимое состояние объекта** — недопустимая комбинация значений (состояний) членов класса и / или состояний базовых классов

Инвариант класса *

- **Инвариант класса** — утверждение, определяющее непротиворечивое состояние объектов этого класса
- **Нарушение инварианта класса** — объект класса имеет некорректное состояние
- **Способы контроля инварианта класса** — проверка инварианта в случаях, когда объект мог изменить своё состояние
  - см. «контрактное программирование», Бертран Маер

Нарушение инварианта класса *

- Приводит к нарушению целостности кода, который использует данный класс
- Приводит к недопустимому состоянию классов, которые его используют
- А значит к нарушению работы всей программы

Контроль инварианта класса на ЯП D

```
class Date {
  int day;
  int hour;

  invariant() {
    assert(1 <= day && day <= 31);
    assert(0 <= hour && hour < 24);
  }
}
```

- Метод `invariant` вызывается каждый раз при потенциальном изменении состояния объекта
- (В чем отличие от C++?)

Вариативность состояния класса *

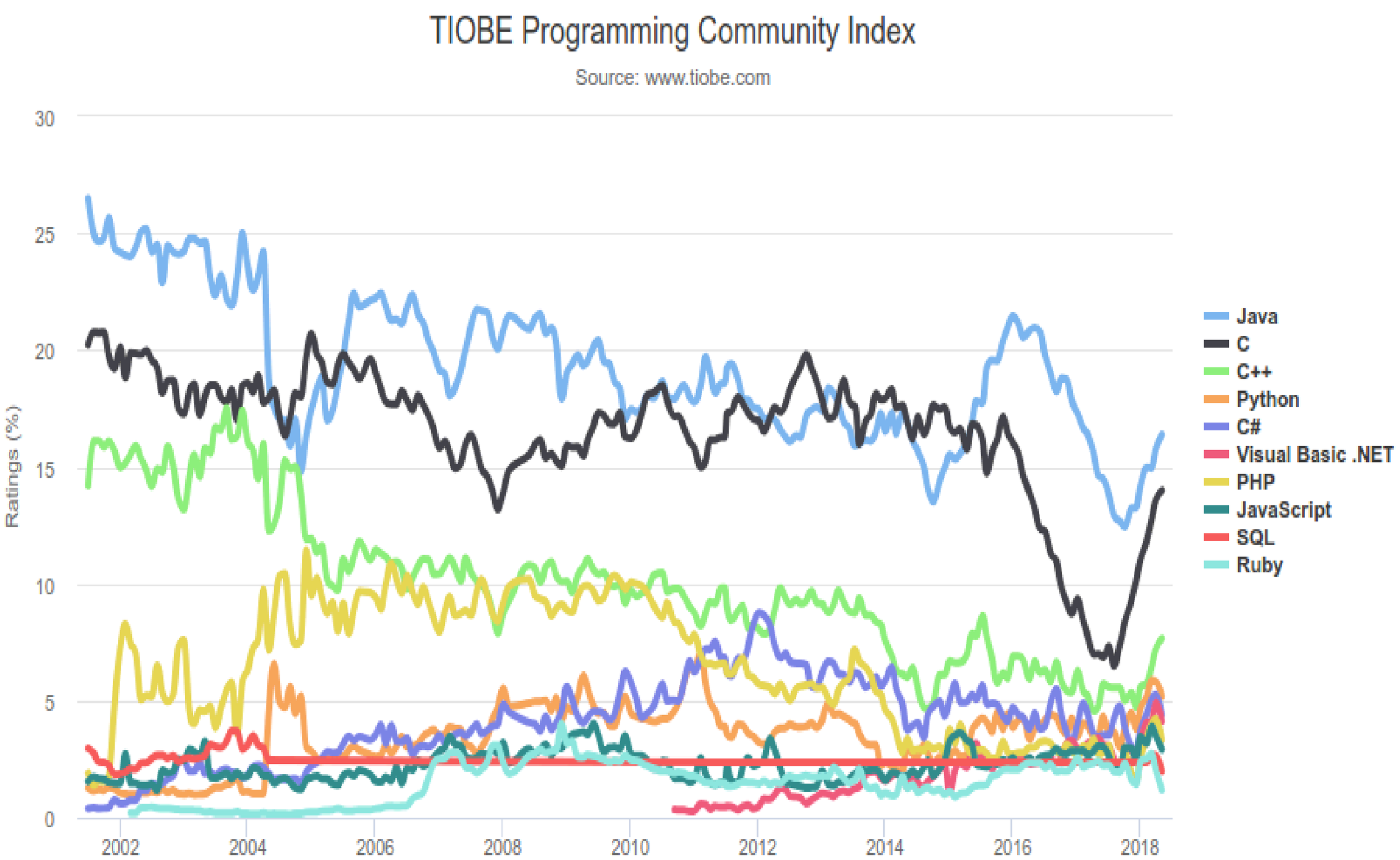
- способность объекта заданного класса изменять своё состояние без нарушения инварианта при выполнении стандартных операций над объектами (копирование, параллельный доступ и т.д.)

Типы вариативности состояний класса *

- **Неизменяемый (immutable)** — объект никогда не изменяет своего состояния
  - для таких объектов инвариант достаточно проверить в конструкторе
- **Копируемый** — при копировании объект не нарушает своего состояния:
  - при создании (конструктор копирования),
  - при передаче в качестве параметра методу (конструктор копирования),
  - при присваивании (оператор присваивания).
- **Некопируемый** — объект может быть только переносим без нарушения своего состояния

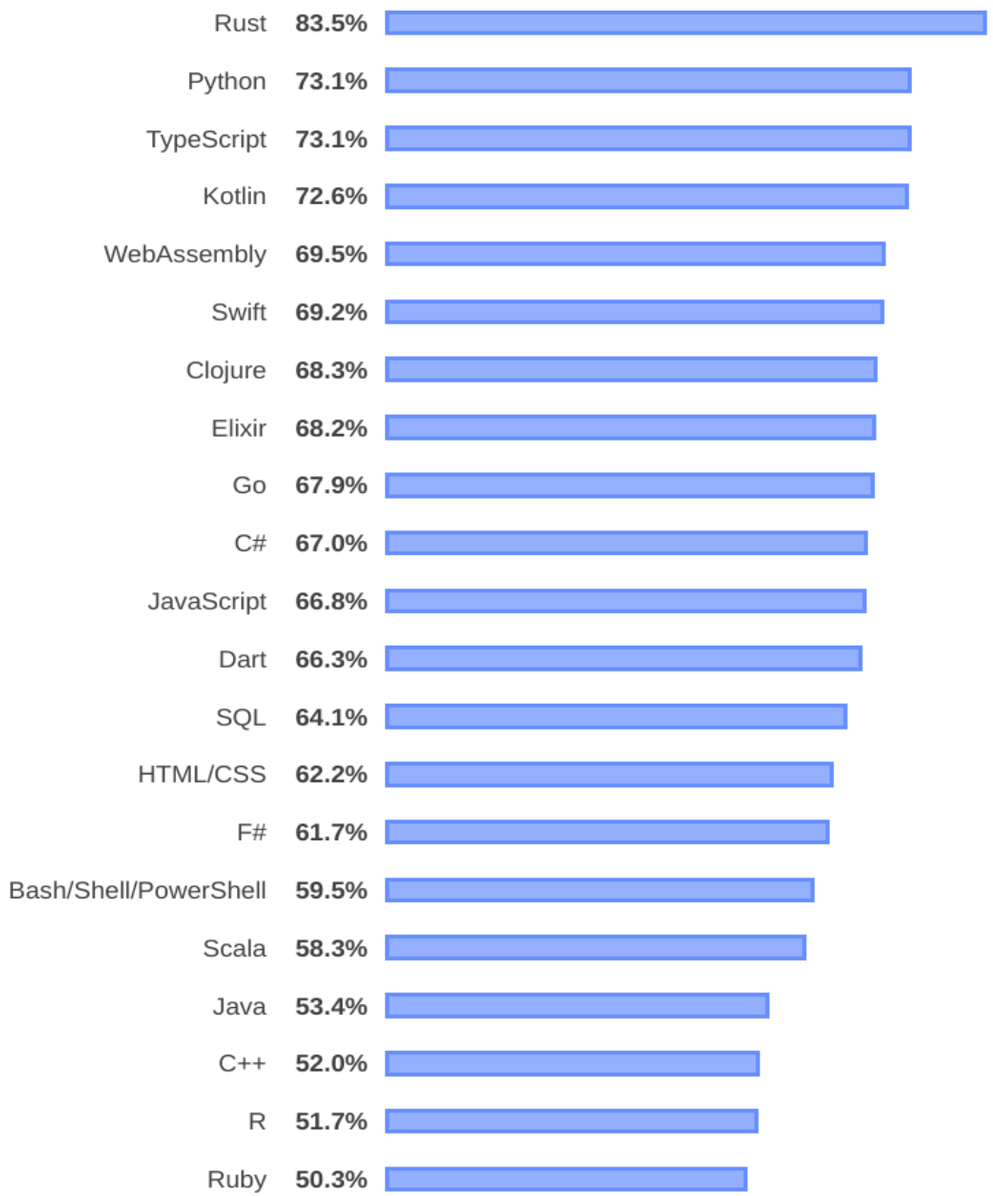
Вопросы?

Петров Антон Александрович
petrov.a@kubsau.ru


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
Java
C
C++
Python
C#
Visual Basic .NET
PHP
JavaScript
SQL
Ruby

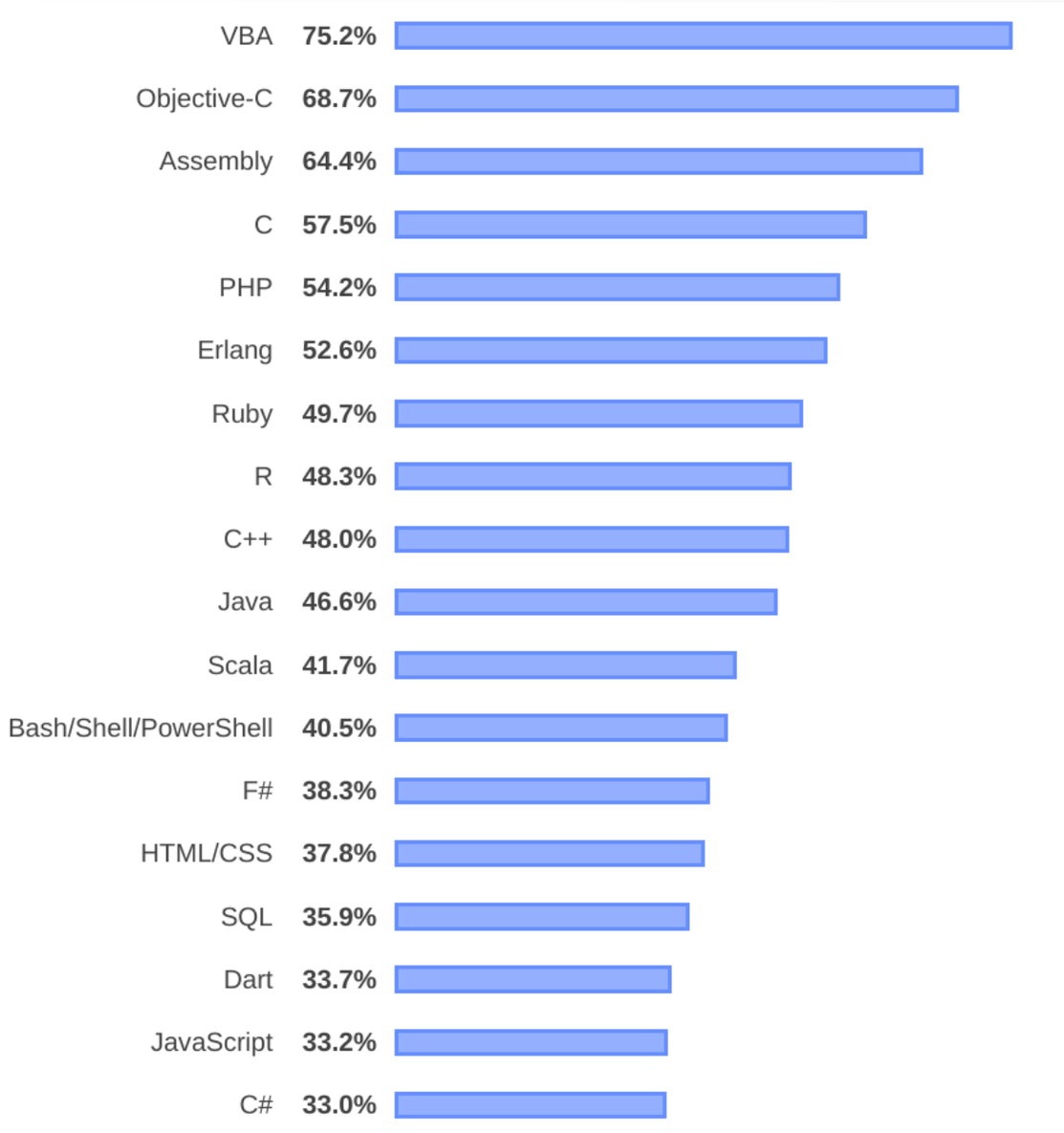
Stack Overflow Developer Survey Results 2019 (https://insights.stackoverflow.com/survey/2019)

- Most Loved Languages



Stack Overflow Developer Survey Results 2019 (https://insights.stackoverflow.com/survey/2019)

- Most Dreaded Languages



Stack Overflow Developer Survey Results 2019 (https://insights.stackoverflow.com/survey/2019)

- Most Wanted Languages