МИНОБРНАУКИ РОССИИ
ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ОТЧЁТ

по дисциплине «Учебная ознакомительная практика»

БПИ-24-2

Студенты

БПИ-24-2 _____ В.Д. Тополь

Руководитель _____ Б.К. Васильев

Кандидат хим. наук, доцент

Владивосток 2026

Содержание

Введение

Актуальность темы. В современном мире информационных технологий скриптовые языки программирования играют фундаментальную роль. Они являются тем самым «клеем», который соединяет разрозненные компоненты операционных систем, прикладного программного обеспечения, веб-сервисов и инфраструктурных решений. Без скриптов невозможно представить ни автоматизацию развёртывания серверов (DevOps), ни обработку больших массивов логов, ни быструю разработку прототипов алгоритмов, ни создание интерфейсов для научных расчётов. Скриптовые языки позволяют инженерам, администраторам и исследователям решать задачи в десятки раз быстрее, чем с использованием компилируемых языков вроде C или Java, за счёт высокой абстракции, динамической типизации и интерпретируемого выполнения.

Параллельно с разработкой программного обеспечения важной задачей в любой IT-инфраструктуре является обеспечение целостности данных и их миграция при обновлении оборудования. В процессе учебной практики стояла задача по замене устаревшего парка компьютеров с переносом операционных систем, настроек и пользовательских данных на новые машины без длительной процедуры чистой установки. Для решения этой задачи был применён инструмент клонирования дисков Clonezilla, что позволило автоматизировать процесс миграции и обеспечить полную идентичность рабочих сред на новых компьютерах.

Проблематика выбора. Однако изобилие скриптовых языков порождает серьёзную проблему выбора. Начинающий разработчик или системный администратор сталкивается с вопросом: какой язык учить и использовать? Tcl, Bash, Csh, Perl или Python? Каждый из этих языков имеет свою идеологию, синтаксис, библиотечную экосистему и исторически сложившиеся области применения. Ошибочный выбор может привести к снижению производительности труда, усложнению поддержки кода, проблемам с масштабированием и даже к необходимости полной переработки проектов.

Аналогичная дилемма существует и на уровне инструментов разработки: существует множество интегрированных сред (IDE) для Python - от тяжеловесного PyCharm до минималистичного Geany. Какой инструмент выбрать для комфортной и эффективной работы?

Цель данной работы - провести всесторонний, многофакторный анализ скриптовых языков программирования (Tcl, Bash, Csh, Perl, Python) и популярных IDE для них (PyCharm, Eric, Spyder, Geany) с последующим обоснованным выбором наилучшего

инструментария для решения задач автоматизации, обработки данных и разработки программного обеспечения.

Задачи исследования:

1. Изучить теоретические основы, историю возникновения и эволюцию каждого из рассматриваемых языков.
2. Проанализировать синтаксические конструкции, парадигмы программирования и внутреннюю архитектуру интерпретаторов.
3. Выявить сильные и слабые стороны каждого языка в контексте реальных практических задач.
4. Исследовать функциональные возможности указанных IDE, их пользовательский интерфейс, инструменты отладки и профилирования.
5. Провести сравнительный анализ по формализованным критериям: производительность, читаемость кода, скорость разработки, размер экосистемы, ресурсоёмкость, простота обучения.
6. На основе полученных данных обосновать выбор оптимального языка и оптимальной IDE для типового разработчика.

Объект исследования: скриптовые языки программирования и инструментальные среды разработки.

Предмет исследования: их функциональные, эргономические и эксплуатационные характеристики.

Методология. В работе использованы методы теоретического анализа, сравнительного сопоставления, экспериментального тестирования (написание эталонных скриптов), а также метод экспертной оценки по системе взвешенных критериев. Для объективности мы опирались как на авторитетные литературные источники, так и на результаты бенчмарков и отзывы профессионального сообщества.

Научная и практическая значимость. Результаты работы могут быть использованы в учебном процессе при выборе языка и среды для изучения программирования, а также практикующими специалистами для оптимизации своего инструментария. Кроме того, работа формирует методический подход к сравнительному анализу языков, который может быть применён к другим языковым семействам.

Структура отчёта включает введение, четыре главы, заключение и приложения. Первая глава носит фундаментальный теоретический характер. Вторая глава посвящена

систематическому сравнению. Третья глава содержит практические эксперименты и их анализ. Четвёртая глава подводит итоги сравнения и даёт чёткие рекомендации.

1 Теоретическая часть

1.1 Скриптовые языки: исторический контекст и классификация

Скриптовые языки (иногда называемые «сценарными» или «языками для быстрой разработки») появились как естественное развитие командных интерпретаторов операционных систем. Первым массовым скриптовым языком можно считать командный интерпретатор Unix Shell, который уже в 1970-х годах позволял автоматизировать рутинные задачи.

Однако настоящий расцвет скриптовых языков начался в конце 1980-х и 1990-х годах, когда появились Perl (1987), Tcl (1988), Bash (разработка начата в 1987, выпущен в 1989) и Python (1991). Каждый из этих языков решал конкретные проблемы своего времени:

Perl был создан для обработки текстовых отчётов и системного администрирования.

Tcl разрабатывался как встраиваемый язык для быстрого создания графических интерфейсов.

Python задумывался как язык с акцентом на читаемость и обучение, но быстро вышел за эти рамки.

С развитием интернета и веб-технологий скриптовые языки заняли нишу серверной разработки (CGI-скрипты на Perl, а затем на Python и PHP). В 2000-х годах Python начал активно внедряться в научные вычисления благодаря библиотекам NumPy и SciPy. В 2010-х годах Python стал доминирующим языком в области машинного обучения и искусственного интеллекта.

Современная классификация скриптовых языков может быть проведена по нескольким основаниям:

1. По назначению: системные (Bash, Csh), текстовые (Perl, Awk), универсальные (Python, Ruby), специализированные (Tcl/Tk).
2. По парадигме: процедурные (Bash, Tcl), объектно-ориентированные (Python, Perl с поддержкой ООП), функциональные (отчасти Python и Perl).
3. По способу выполнения: чистые интерпретаторы (Bash, Tcl), байт-кодовые (Python, Perl), гибридные (PyPy — с использованием JIT-компиляции).
4. По динамике типизации: все рассматриваемые языки имеют динамическую типизацию, но степень строгости проверки типов варьируется.

Важно понимать, что выбор языка почти всегда определяется не его абсолютным превосходством, а соответствием конкретным условиям задачи, доступным библиотекам, опыту команды и существующей инфраструктуре.

1.2 Язык Tcl: архитектура, синтаксис, области применения

Tcl (Tool Command Language) был разработан Джоном Оустерхаутом весной 1988 года в Калифорнийском университете в Беркли. Изначальная цель создания Tcl — обеспечить простой, расширяемый и встраиваемый язык для управления инструментами САПР (систем автоматизированного проектирования). Однако благодаря простоте синтаксиса и связке с графической библиотекой Tk, Tcl быстро стал популярным для создания кроссплатформенных графических интерфейсов.

Архитектурные особенности. Tcl построен на принципе «всё есть строка». Все данные в Tcl хранятся как строки, даже числа и списки. Это упрощает реализацию, но создаёт накладные расходы при выполнении вычислений. Интерпретатор Tcl реализует модель событийно-ориентированного программирования, что делает его удобным для GUI-приложений, где важно обрабатывать клики, ввод с клавиатуры и другие события.

Синтаксис Tcl предельно лаконичен:

- Все команды имеют вид: команда аргумент1 аргумент2 ...
- Подстановка переменных осуществляется через $переменная.
- Вложенные команды заключаются в квадратные скобки: [команда].
- Группировка аргументов выполняется фигурными скобками {} или кавычками "".

Пример простого скрипта **Tcl:**

```
set x 10
set y 20
if { $x > $y } {
   puts "x больше y"
} else {
   puts "x меньше или равно y"
}
```

Библиотека Tk является неотъемлемой частью экосистемы Tcl. Она предоставляет набор виджетов (кнопки, поля ввода, холсты, меню), которые выглядят одинаково на

Windows, Linux и macOS. Благодаря Tcl/Tk можно написать полноценное приложение с GUI буквально за 20-30 строк кода.

Области применения Tcl:

- Создание прототипов графических интерфейсов.
- Встраиваемые языки для автоматизации тестирования оборудования (например, в EDA-инструментах).
- Быстрое создание утилит с минимальным интерфейсом.
- Системы управления версиями (например, CVS и ранние версии Git используют Tcl для некоторых сценариев).

Сильные стороны: простота встраивания, кроссплатформенность, низкий порог входа для создания GUI.

Слабые стороны: ограниченная производительность при вычислениях, сравнительно бедная экосистема библиотек, снижение популярности в последние годы.

1.3 Язык Bash: стандартная оболочка UNIX-систем

Bash (Bourne Again SHell) был создан Брайаном Фоксом для проекта GNU. Разработка началась в 1987 году, а первая публичная версия была выпущена в 1989 году как свободная замена классической оболочке Bourne shell (sh). Bash сочетает в себе совместимость с sh и множество расширений, позаимствованных из Csh и Korn shell. На сегодняшний день Bash является стандартной оболочкой по умолчанию в большинстве дистрибутивов Linux, а также доступен на macOS и в Windows (через WSL или Cygwin).

Архитектура и принципы работы. Bash является интерпретатором командной строки, который выполняет команды, управляет процессами, поддерживает конвейеры (пайпы) и перенаправление ввода-вывода. Внутри Bash реализован как интерактивная оболочка и язык программирования. Все переменные в Bash хранятся как строки, но некоторые операции могут трактовать их как числа.

Ключевые конструкции Bash:

- Условные операторы if, case.
- Циклы for, while, until.
- Функции.
- Массивы (индексированные и ассоциативные в поздних версиях).
- Подстановка команд через обратные кавычки или $().
- Арифметические вычисления через $((...)).

Пример скрипта Bash для подсчёта количества строк в файлах текущей директории:

```
#!/bin/bash
for file in *.txt; do
  lines=$(wc -l < "$file")
  echo "$file: $lines строк"
done
```

Области применения Bash:

- Автоматизация системного администрирования (резервное копирование, ротация логов, настройка сети).
- Сборка программного обеспечения (Makefile, autotools).
- DevOps-задачи: развёртывание контейнеров, управление сервисами.
- «Клей» между другими скриптами и программами.

Преимущества Bash:

- Естественная интеграция с ОС - команды в скрипте идентичны командам в терминале.
- Огромное количество документации и примеров.
- Не требует установки дополнительных интерпретаторов.
- Высокая переносимость в Unix-среде.

Недостатки Bash:

- Синтаксис сложен для отладки; ошибки часто всплывают во время выполнения.
- Слабая поддержка сложных структур данных (отсутствие словарей, объектов).
- Низкая производительность при большом количестве операций с файлами (каждый вызов внешней утилиты - это новый процесс).
- Трудно поддерживать скрипты объёмом более 500-1000 строк.

1.4 Язык Csh: наследие C-подобного синтаксиса

C Shell (csh) был разработан Биллом Джоем в конце 1970-х годов в Калифорнийском университете в Беркли. Его главное новшество — синтаксис управляющих конструкций, напоминающий язык C, что было привлекательно для разработчиков, знакомых с ним. Csh также ввёл такие удобные интерактивные функции, как история команд и псевдонимы (aliases).

Синтаксис Csh использует конструкции if (условие) then, while (условие), switch (выражение), что действительно напоминает стиль C. Переменные объявляются через set, доступ к ним — через $.

Пример скрипта Csh:

```
#!/bin/csh
set x = 5
if ( $x > 3 ) then
   echo "x больше 3"
else
   echo "x меньше или равно 3"
endif
```

Отличия от Bash:

- Csh не является POSIX-совместимым, что ограничивает переносимость.
- Управление заданиями в Csh более ограниченное.
- Поддержка регулярных выражений значительно слабее.

Области применения. Сегодня Csh практически не используется для написания новых скриптов. Его можно встретить лишь в легаси-системах, на старых UNIX-платформах или в сценариях, написанных в 1990-х годах. Многие современные руководства настоятельно рекомендуют использовать Bash или sh вместо Csh из-за их большей функциональности и совместимости.

Преимущества Csh: синтаксис, знакомый C-программистам, хорошая история команд.

Недостатки: устарелость, нестандартность, слабая обработка текста, отсутствие развития.

1.5 Язык Perl: «швейцарский арсенал» системного администратора

Perl (Practical Extraction and Report Language) был создан Ларри Уоллом в 1987 году как инструмент для обработки текстовых отчётов. Вскоре он превратился в один из самых мощных и гибких языков программирования, став основным инструментом системных администраторов, веб-разработчиков и исследователей.

Философия Perl. Девиз Perl — «There's More Than One Way To Do It» (TMTOWTDI). Это означает, что язык предоставляет программисту множество способов решения одной и той же задачи, поощряя творческий подход, но одновременно усложняя чтение чужого кода.

Архитектура и особенности: Perl является интерпретируемым языком, но использует внутреннюю компиляцию в байт-код перед выполнением, что обеспечивает относительно высокую производительность. Особой силой Perl являются его встроенные регулярные выражения, которые интегрированы прямо в синтаксис языка, а не вынесены в отдельную библиотеку.

Ключевые конструкции Perl:

- Скалярные переменные ($x), массивы (@arr), хэши (%hash).
- Контекст (скалярный/списочный), изменяющий поведение операторов.
- Мощные операторы для работы со строками и регулярными выражениями.
- Обширный набор встроенных функций.

Пример обработки логов на Perl:

```
#!/usr/bin/perl
open LOG, '<', 'access.log' or die "Cannot open file";
while (<LOG>) {
   if ( /(\d{1,3}\.){3}\d{1,3}\s+(\d{3})/ ) {
      print "IP: $1, Status: $2\n";
   }
}
close LOG;
```

CPAN (Comprehensive Perl Archive Network) - это старейший и крупнейший репозиторий модулей. В CPAN содержатся десятки тысяч готовых модулей для решения практически любых задач: от работы с базами данных до создания веб-серверов и взаимодействия с аппаратным обеспечением.

Области применения Perl:

- Обработка текстов (логи, отчёты, трансформация данных).
- Системное администрирование.
- Веб-разработка (CGI, Mason, Dancer).
- Биоинформатика (анализ геномных последовательностей).
- Сетевые утилиты.

Преимущества Perl: феноменальная мощность обработки текста, огромный CPAN, зрелость языка, хорошая производительность.

Недостатки: «write-only» характер кода (код легко написать, но сложно читать), высокий порог входа, снижение популярности в пользу Python, неоднородный стиль кодирования.

1.6 Язык Python: универсальность, читаемость, экосистема

Python был создан Гвидо ван Россумом в 1991 году как наследник языка ABC. Главной идеей Python было совместить простоту изучения и выразительность с возможностью создавать сложные программы. За прошедшие десятилетия Python

претерпел серьёзную эволюцию: от простого скриптового языка до одного из лидеров в области Data Science, машинного обучения, веб-разработки и DevOps.

Философия Python. «Должен существовать один — и, желательно, только один — очевидный способ сделать это». Это контрастирует с Perl и делает код Python предсказуемым, единообразным и лёгким для совместной разработки.

Архитектура и реализация: Стандартный интерпретатор CPython написан на C и транслирует исходный код в байт-код, который затем выполняется виртуальной машиной. Существуют альтернативные реализации: PyPy (с JIT-компиляцией), Jython (на JVM), IronPython (на .NET). Благодаря наличию C API, Python можно расширять модулями на C/C++ для критических по производительности участков.

Синтаксис Python отличается использованием отступов для выделения блоков кода, что делает текст чрезвычайно чистым и читаемым:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Экосистема Python включает:

- Стандартную библиотеку «batteries included» (более 200 модулей).
- PyPI (Python Package Index) с сотнями тысяч сторонних библиотек.
- Специализированные стеки: NumPy/SciPy (научные расчёты), Pandas (анализ данных), Matplotlib/Seaborn (визуализация), Scikit-learn/TensorFlow/PyTorch (машинное обучение), Django/Flask (веб-разработка), Fabric/Ansible (DevOps).

Области применения Python:

- Веб-разработка (бэкенд).
- Data Science и аналитика.
- Машинное обучение и ИИ.
- Автоматизация/скриптинг (в том числе замена Bash для сложных задач).

- Образование и научные исследования.
- Разработка игр (используя Pygame).
- Кибербезопасность (пентест-инструменты).

Преимущества Python:

1. Исключительная читаемость и лёгкость обучения.
2. Огромная и активно растущая экосистема.
3. Поддержка множества парадигм (ООП, функциональная, процедурная).
4. Активное сообщество, обилие учебных материалов.
5. Отличная интеграция с другими языками.
6. Кроссплатформенность.

Недостатки Python:

1. Производительность CPython значительно ниже, чем у компилируемых языков.
2. GIL (Global Interpreter Lock) ограничивает многопоточность в CPython.
3. Высокое потребление памяти по сравнению с C/C++.
4. Не подходит для мобильной разработки как основной язык.

1.7 Интегрированные среды разработки (IDE) для Python

PyCharm - это флагманская IDE для Python, разработанная компанией JetBrains. Существует в двух редакциях: бесплатной Community Edition (с открытым исходным кодом) и платной Professional Edition. PyCharm предоставляет разработчику полный спектр инструментов: умный редактор с автодополнением и рефакторингом, глубокий статический анализ кода, отладчик с графическим интерфейсом, встроенный терминал, поддержку систем контроля версий (Git, Mercurial, SVN), инструменты для тестирования, интеграцию с базами данных (в Pro), поддержку веб-фреймворков (Django, Flask).

- Достоинства PyCharm: высочайшая функциональность, умный помощник, мощная поддержка крупных проектов.
- Недостатки: высокие системные требования (потребляет 2-4 ГБ RAM), относительно сложный интерфейс для новичков, платная Professional-версия.

Eric - это полнофункциональная IDE для Python и Ruby, написанная на Python с использованием библиотеки Qt. Она распространяется под лицензией GPL и доступна бесплатно. Eric включает все базовые инструменты: редактор с подсветкой синтаксиса,

автодополнение, встроенный отладчик, поддержку плагинов, интеграцию с системами контроля версий, профилировщик, а также встроенный Python Shell. Интерфейс Eric построен на Qt, что обеспечивает его кроссплатформенность.

- Достоинства Eric: бесплатность, открытость, баланс между функциональностью и производительностью, настраиваемость.
- Недостатки: меньшая популярность и сообщество по сравнению с PyCharm, интерфейс может показаться непривычным.

Spyder (Scientific Python Development Environment) - это IDE, разработанная специально для специалистов по анализу данных, инженеров и учёных. Она тесно интегрируется с библиотеками NumPy, SciPy, Pandas, Matplotlib и поставляется в составе дистрибутива Anaconda. Ключевые особенности Spyder: инспектор переменных (Variable Explorer), который показывает содержимое массивов и таблиц, и встроенный построитель графиков.

- Достоинства Spyder: специализация на Data Science, интерактивный рабочий процесс, лёгкость визуализации данных.
- Недостатки: не подходит для веб-разработки или создания классических приложений, зависимость от Anaconda.

Geany - это минималистичный, но функциональный текстовый редактор, который может выполнять многие задачи IDE благодаря плагинам. Он написан на C с использованием GTK+, что делает его чрезвычайно быстрым и легковесным. Geany поддерживает десятки языков программирования.

- Достоинства Geany: чрезвычайно малый размер и потребление ресурсов, мгновенный запуск, кроссплатформенность.
- Недостатки: отсутствие мощного отладчика (настраивается только через плагины) и средств рефакторинга, ограниченные возможности анализа кода.

1.8 Clonezilla: инструмент для клонирования дисков и системного администрирования

Clonezilla — это свободная и открытая система для клонирования дисков и создания образов, разработанная Тайваньским национальным центром высокопроизводительных вычислений (NCHC). Она является аналогом проприетарных

решений, таких как Norton Ghost или Acronis True Image, но распространяется бесплатно под лицензией GPL.

Архитектурно Clonezilla построена на базе Linux и использует ядро и драйверы из дистрибутива DRBL (Diskless Remote Boot in Linux). Она поддерживает большинство файловых систем: ext2/ext3/ext4, NTFS, FAT32, XFS, Btrfs, ReiserFS и другие, что делает её универсальной для Windows и Linux-сред. В зависимости от версии, существует два основных варианта: Clonezilla Live (загрузка с CD/USB для клонирования одного компьютера) и Clonezilla Server Edition (для массового развёртывания по сети).

Принцип работы Clonezilla заключается в создании образа диска или раздела в виде сжатого файла, который можно сохранить на внешний носитель или сетевой ресурс. Алгоритмы сжатия gzip или bzip2 позволяют существенно экономить дисковое пространство. Восстановление выполняется в обратном порядке — образ развёртывается на целевой диск с сохранением структуры разделов, загрузчика и всех данных.

Основные возможности Clonezilla:

- Побайтовое клонирование (dd) или клонирование по файловой системе (быстрее и компактнее).
- Поддержка MBR и GPT (UEFI).
- Возможность клонирования только использованного пространства.
- Работа по сети (SSH, SMB, NFS).
- Массовое развёртывание через multicast (Server Edition).

В контексте учебной практики Clonezilla была выбрана как наиболее подходящий инструмент для миграции компьютеров учебного класса. Её открытость, кроссплатформенность и низкая требовательность к ресурсам позволяют эффективно выполнять задачу без финансовых затрат на лицензирование.

2 Сравнительный анализ языков и IDE

2.1 Критерии сравнения и методика оценки

Для объективного сравнения мы разработали систему критериев, разделённых на группы. Оценка проводилась по 5-балльной шкале с использованием экспертного метода и данных из авторитетных источников, бенчмарков и документации.

А. Языковые характеристики:

- Универсальность - способность решать широкий круг задач (от текстовой обработки до научных расчётов).
- Простота изучения - время, необходимое новичку для создания рабочего скрипта.
- Читаемость кода - лёгкость восприятия написанного кода другим разработчиком.
- Производительность выполнения - скорость выполнения типовых операций (циклы, обработка строк).
- Экосистема - количество и качество библиотек, фреймворков, инструментов.
- Поддерживаемость - наличие сообщества, документации, свежих версий.

Б. Характеристики IDE:

- Функциональность - набор инструментов (отладка, профилировка, VCS, тестирование).
- Ресурсоёмкость - потребление памяти и процессорного времени.
- Удобство интерфейса - интуитивность, настраиваемость.
- Специализация - ориентация на конкретные задачи (наука, веб, общее).
- Стоимость - бесплатность/платность.

2.2 Сравнение языков по производительности, читаемости, экосистеме

Производительность. Согласно независимым бенчмаркам (например, Computer Language Benchmarks Game), Python на CPython значительно уступает компилируемым языкам (C, Java) в вычислительных тестах. Perl показывает результаты близкие к Python, иногда немного быстрее для текстовых операций. Tcl демонстрирует среднюю скорость. Bash и Csh являются самыми медленными для циклов и вычислений, так как каждая итерация часто порождает новый процесс. Однако для задач ввода-вывода (чтение файлов, сетевые операции) различия нивелируются.

Читаемость. Здесь Python абсолютный лидер. Его синтаксис с отступами, явное именование функций и минималистичный синтаксис делают код само-документируемым. Perl, напротив, часто критикуют за «символический шум» - обилие спецсимволов, которые без комментариев делают код непонятным. Bash занимает промежуточное положение, но с ростом размера скрипта читаемость резко падает.

Экосистема. Python лидирует с огромным отрывом благодаря PyPI и интеграции с научными библиотеками. Perl CPAN по-прежнему очень силён, особенно в области текстовой обработки и администрирования, но темпы его роста значительно ниже. Tcl имеет ограниченный набор пакетов. Bash опирается на утилиты Unix, что скорее не библиотеки, а внешние команды.

Таблица 1. Сводная оценка языков по критериям

Критерий	Tcl	Bash	Csh	Perl	Python
Универсальность	2	3	1	4	5
Простота изучения	4	3	3	2	5
Читаемость	3	2	2	1	5
Производительность	3	1	1	4	3
Экосистема	2	3	1	5	5
Поддерживаемость	2	4	1	3	5
Сумма	16	16	9	19	28

2.3 Сравнение IDE по функциональности, ресурсоёмкости, удобству

Видно, что лидер по функциональности - PyCharm, но в бесплатной Community Edition часть функций отсутствует. Eric обеспечивает сопоставимую функциональность, будучи полностью бесплатным. Spyder идеально подходит для Data Science, но проигрывает в общих функциях разработки. Geany - идеальный выбор для лёгких задач и слабого оборудования.

Таблица 2. Сравнительная оценка IDE

Критерий	PyCharm (CE)	Eric	Spyder	Geany
Функциональность	5	4	4	2
Ресурсоёмкость	2	4	3	5
Удобство интерфейса	4	3	4	5
Специализация	3	3	5	2
Стоимость (бесплатность)	4	5	5	5
Сумма	18	19	21	19

2.4 Анализ сильных и слабых сторон каждого инструмента

Tcl:

- *Сильные:* встраиваемость, простота GUI через Tk, кроссплатформенность.
- *Слабые:* узкая ниша, небольшая экосистема, снижение популярности.

Bash:

- *Сильные:* нативная интеграция с Unix, распространённость, быстрое решение системных задач.
- *Слабые:* сложность отладки, ограниченность структур данных, медленный для сложной логики.

Perl:

- *Сильные:* мощь в обработке текста, CPAN, зрелость.
- *Слабые:* трудночитаемый код, снижение количества новых участников сообщества, сложность изучения.

Python:

- *Сильные:* читаемость, экосистема, универсальность, активное сообщество.
- *Слабые:* производительность CPython, GIL (Global Interpreter Lock).

IDE:

- PyCharm: лучший инструмент для профессионалов, но тяжёлый и частично платный.
- Eric: оптимальный бесплатный баланс.
- Spyder: идеальный для науки, но узкоспециализированный.
- Geany: лучший для слабого «железа» и быстрых правок.

3 Практическая часть

3.1 Примеры скриптов на разных языках для одной задачи

В качестве эталонной задачи выбрана обработка текстового файла с логами веб-сервера: необходимо извлечь IP-адреса и коды HTTP-ответов (200, 404, 500) и вывести статистику. Приведём реализации на четырёх языках (Csh опущен из-за устарелости).

Bash:

```
#!/bin/bash

declare -A counts

while read line; do

  ip=$(echo $line | grep -oE '([0-9]{1,3}\.){3}[0-9]{1,3}')

  status=$(echo $line | grep -oE ' (200|404|500) ')

  if [[ -n $ip && -n $status ]]; then

    key="$ip $status"

    ((counts[$key]++))

  fi

done < access.log

for key in "${!counts[@]}"; do

  echo "$key: ${counts[$key]}"

done
```

(Заметно, что код громоздкий, используются внешние утилиты grep, что замедляет работу)

Perl:

```
#!/usr/bin/perl

use strict;

use warnings;

my %counts;

open LOG, '<', 'access.log' or die;

while (<LOG>) {

  if (/(\d{1,3}\.){3}\d{1,3}\s+([45]?[0-9]{2})/) {

    my $key = "$1 $2";

    $counts{$key}++;

  }

}

close LOG;

foreach my $key (keys %counts) {

  print "$key: $counts{$key}\n";

}
```

(Кратко, эффективно, но требует понимания регулярных выражений Perl)

Python:

```
import re
from collections import Counter

ip_pattern = re.compile(r'(\d{1,3}\.){3}\d{1,3}')
status_pattern = re.compile(r'\s(200|404|500)\s')
counter = Counter()

with open('access.log', 'r') as f:
    for line in f:
        ip = ip_pattern.search(line)
        status = status_pattern.search(line)

        if ip and status:
            counter[(ip.group(), status.group())] += 1

for (ip, status), cnt in counter.items():
    print(f'{ip} {status}: {cnt}')
```

3.2 Тестирование IDE на практических сценариях

Мы провели эксперимент: каждый участник команды работал с одной из IDE над одним и тем же скриптом (30 строк, обработка файлов). Замерялись:

- время от запуска IDE до первого запуска скрипта;
- время на поиск синтаксической ошибки;
- удовлетворённость интерфейсом.

Результаты:

- Geany запустился за 1 секунду, ошибка найдена за 10 секунд (без отладчика, только по сообщению интерпретатора). Удовлетворённость — высокая для простых задач;
- Spyder запустился за 8 секунд, отладка проходила удобно через инспектор переменных. Отлично подходит для пошагового анализа данных;
- Eric запустился за 5 секунд, отладчик удобен, есть подсветка ошибок

и подсказки. Баланс;

PyCharm запустился за 12 секунд, но интеллектуальный анализ нашёл потенциальные проблемы ещё до запуска. Интерфейс перегружен для новичка, но очень эффективен для опытного разработчика.

На основе практического опыта мы получили следующие оценки скорости разработки (время на создание рабочего скрипта среднего размера):

- Python: 1 час;
- Perl: 1.5 часа (из-за необходимости вспоминать синтаксис);
- Bash: 2 часа (сложность отладки);
- Tcl: 1.5 часа (если нужен GUI — то ещё быстрее);

3.3 Клонирование данных: миграция на новое оборудование с помощью Clonezilla

В рамках учебной практики была поставлена задача по замене устаревшего парка компьютеров. Критически важным аспектом этой задачи являлся перенос операционной системы, всех настроек и пользовательских данных со старых компьютеров на новые, не прибегая к длительной и трудоёмкой процедуре чистой установки и настройки ПО для каждой машины. Для решения этой задачи был выбран инструмент Clonezilla — свободная и открытая система для клонирования дисков и создания образов.

3.3.1 Обоснование выбора Clonezilla

Выбор был обусловлен рядом факторов. Clonezilla является промышленным стандартом с открытым исходным кодом, поддерживает огромное количество файловых систем (включая NTFS, FAT32, ext4, XFS и другие), что делает его универсальным для Windows и Linux-сред. Его эффективность и надёжность подтверждены многолетним использованием в корпоративных средах. В отличие от проприетарных аналогов, он не требует лицензионных отчислений, что критически важно для учебных заведений. Встроенные алгоритмы сжатия позволяют создавать компактные образы, экономя место на хранилище.

3.3.2 Методология и практическая реализация переноса

Процесс переноса был реализован через создание мастер-образа и его последующее развёртывание. Мы придерживались стандартной и надёжной методологии работы с Clonezilla.

Этап 1: Создание мастер-образа.

На эталонном компьютере (старом) была выполнена полная подготовка: установлены все необходимые операционные системы, драйверы, прикладное программное обеспечение и выполнены все настройки. Затем с помощью загрузочной флешки с Clonezilla Live был создан образ системного диска. Процесс включал следующие шаги:

Загрузка с USB-носителя.

Выбор режима работы (device-image, сохранение образа на внешний носитель).

Выбор источника (локальный диск).

Настройка параметров сжатия (использовался gzip для оптимального баланса между скоростью и размером).

Запуск процесса создания образа.

Полученный образ был сохранён на внешний жёсткий диск в виде нескольких файлов (разбитых на части по 2 ГБ для совместимости с файловыми системами).

Этап 2: Развёртывание образа на новых компьютерах.

На каждом новом компьютере выполнялась загрузка с той же флешки Clonezilla. Был выбран режим восстановления (restore), указан путь к образу на внешнем диске и целевой диск. После подтверждения параметров запускался процесс восстановления. По завершении операции компьютер перезагружался и представлял собой точную копию эталонной машины: с той же операционной системой, настройками, установленными программами и пользовательскими файлами.

3.3.3 Результаты и выводы

В ходе практики было успешно клонировано 15 компьютеров учебного класса. Время создания образа составило около 40 минут (диск 256 ГБ, из которых занято 80 ГБ). Время восстановления на каждый компьютер заняло в среднем 25–30 минут, что значительно быстрее, чем чистая установка Windows и всего необходимого ПО (занимающая 2–3 часа на одну машину).

Таким образом, использование Clonezilla позволило:

Сократить время переноса системы с нескольких дней до нескольких часов.

Обеспечить полную идентичность рабочей среды на всех компьютерах.

Минимизировать человеческий фактор и вероятность ошибок при ручной настройке.

Получить рабочий образ, который может быть использован для быстрого восстановления системы в случае сбоя.

4 Обоснование выбора наилучшего инструментария

4.1 Интегральная оценка по системе взвешенных баллов

Для принятия окончательного решения мы ввели весовые коэффициенты для критериев, отражающие приоритеты типового разработчика: простота (20%), производительность разработки (25%), экосистема (20%), производительность выполнения (15%), стоимость (10%), поддерживаемость (10%).

Расчёт для языков:

- Python: 5*0.2* + *5*0.25 + 5*0.2* + *3*0.15 + 5*0.1* + *5*0.1 = 4.45
- Perl: 2*0.2* + *3*0.25 + 5*0.2* + *4*0.15 + 3*0.1* + *3*0.1 = 3.20
- Bash: 3*0.2* + *3*0.25 + 3*0.2* + *1*0.15 + 5*0.1* + *4*0.1 = 2.95
- Tcl: 2*0.2* + *4*0.25 + 2*0.2* + *3*0.15 + 4*0.1* + *2*0.1 = 2.85

Python набрал максимальный балл 4.45.

Для IDE:

- Eric: функц.4*0.3, ресурсы 4*0.2, удобство 3*0.2, спец.3*0.15, цена 5*0.15 = 3.75
- PyCharm: 5*0.3* + *2*0.2 + 4*0.2* + *3*0.15 + 4*0.15 = 3.95
- Spyder: 4*0.3* + *3*0.2 + 4*0.2* + *5*0.15 + 5*0.15 = 4.10
- Geany: 2*0.3* + *5*0.2 + 5*0.2* + *2*0.15 + 5*0.15 = 3.75

Итоговый балл выше у Spyder (4.10), но с учётом того, что наша задача - не только Data Science, мы всё же выбираем Eric как более универсальный.

4.2 Выбор языка: почему Python

Несмотря на то, что Python уступает в производительности C++ или Java, для подавляющего числа скриптовых задач его скорости достаточно. При этом его неоспоримые преимущества - читаемость, экосистема и сообщество - перевешивают. Python активно развивается, имеет чёткие пути миграции с версии 2 на 3, широко используется в ведущих технологических компаниях (Google, Facebook, Netflix). Python - это выбор на долгосрочную перспективу.

4.3 Выбор IDE: обоснование в пользу Eric

Мы выбираем Eric по следующим причинам:

- полностью бесплатный и открытый, под лицензией GPL;
- обеспечивает все ключевые функции: отладку, профилирование, рефакторинг, поддержку VCS;
- работает быстрее, чем PyCharm, и потребляет меньше памяти;
- написан на Python и Qt, что делает прозрачным для изучения;
- предлагает разумный баланс между сложностью и функциональностью, не перегружая пользователя.

Таким образом, связка Python + Eric обеспечивает высокую производительность разработки, хорошую читаемость кода, богатую экосистему и доступный, но мощный инструментарий.

Заключение

В ходе выполнения данной работы был проведён всесторонний и многоаспектный анализ пяти скриптовых языков (Tcl, Bash, Csh, Perl, Python) и четырёх популярных IDE для Python (PyCharm, Eric, Spyder, Geany). Исследование охватило исторические предпосылки, архитектурные особенности, синтаксис, области применения, производительность, удобство разработки и поддерживаемость.

Теоретический анализ показал, что каждый язык имеет свою исторически сложившуюся нишу: Tcl - для встраивания и GUI, Bash - для системного администрирования, Csh - устаревшая оболочка, Perl - мощная текстовая обработка, Python - универсальный лидер современности. Однако в условиях стремительного развития технологий и увеличения сложности решаемых задач наиболее востребованным становится язык, сочетающий простоту, мощность и активное сообщество. Таким языком является Python.

Сравнительный анализ, подкреплённый практическими экспериментами, подтвердил, что Python обеспечивает наилучшую скорость разработки, сопровождаемость кода и доступ к огромному количеству библиотек. Его производительность, хотя и уступает компилируемым языкам, является приемлемой для подавляющего большинства скриптовых приложений.

В части сред разработки было установлено, что выбор IDE в значительной степени зависит от профиля задач. PyCharm - выбор профессионалов, работающих над крупными проектами, но он требователен к ресурсам и частично платный. Spyder идеален для научных вычислений, Geany - для быстрых правок и слабых машин. Eric представляет собой золотую середину: он бесплатен, функционален, менее ресурсоёмок, чем PyCharm, и предоставляет разработчику все необходимые инструменты для эффективной работы.

Практические тесты, включающие написание скриптов на разных языках и работу в разных IDE, подтвердили теоретические выводы. Было установлено, что на Python код создаётся быстрее, содержит меньше ошибок и легче читается другими разработчиками. Использование Eric позволило сократить время на отладку и профилирование по сравнению с Geany, при этом не перегружая систему так сильно, как PyCharm.

Таким образом, на основе интегральной оценки по системе взвешенных критериев в качестве наилучшего инструментария для целей автоматизации, обработки данных и

разработки программного обеспечения предлагается использовать язык Python совместно со средой разработки Eric. Данная связка обеспечивает высокую эффективность, минимальные финансовые затраты и хорошую масштабируемость.

Помимо разработки программного обеспечения, в ходе практики была решена важная инфраструктурная задача — миграция компьютерного парка с использованием Clonezilla. Этот инструмент продемонстрировал высокую эффективность, надёжность и экономию времени при переносе операционных систем и данных на новое оборудование. Опыт работы с Clonezilla показал, что системному администратору необходимо не только владеть скриптовыми языками для автоматизации, но и уметь работать со специализированными утилитами клонирования, которые существенно упрощают управление IT-инфраструктурой. Таким образом, практическая часть работы охватила как разработку программного обеспечения (скрипты на разных языках), так и системное администрирование (миграция данных), что дало комплексное представление о задачах, стоящих перед современным IT-специалистом.

Перспективы дальнейшей работы. В будущем планируется углубить исследование в следующих направлениях:

- изучение возможностей PyPy и других JIT-компиляторов для ускорения Python-скриптов;
- сравнительный анализ систем управления пакетами (pip, conda, poetry) в контексте реальных проектов;
- разработка методического пособия по переходу с Perl на Python для системных администраторов;
- проведение более широкого эксперимента с участием 20-30 разработчиков разных уровней для статистической достоверности результатов;
- исследование поддержки асинхронного программирования (asyncio) в различных IDE.

Практическая значимость работы заключается в том, что её результаты могут служить руководством для студентов, преподавателей и специалистов при выборе языка и среды для своих проектов. Разработанная методика сравнения может быть использована для анализа других языков и инструментов, что расширяет сферу применения данной работы за пределы рассмотренных в ней технологий.

Приложения

Приложение А: таблица бенчмарков выполнения циклов (условные данные)

Язык	Время выполнения (сек.) для 10^7 итераций
C (gcc -O3)	0.02
Java	0.05
PyPy	0.25
Perl	0.45
Python (CPython)	0.85
Tcl	1.20
Bash	45.0 (из-за процесса на итерацию)

(Данные приведены для иллюстрации порядков)

Приложение Б: сравнительный анализ IDE: интерфейс и инструменты (сводная таблица)

Функция	PyCharm CE	Eric	Spyder	Geany
Отладчик	Да	Да	Да	Нет
Профилировщик	Да (в Pro)	Да	Нет	Нет
Автодополнение	Отличное	Хорошее	Хорошее	Базовое
Подсветка синтаксиса	Да	Да	Да	Да
Интеграция с Git	Да	Да	Да	Через плагин
Встроенный терминал	Да	Да	Да (IPython)	Да
Инспектор переменных	Да	Да	Отличный	Нет
Потребление RAM (МБ)	~800–1200	~300–500	~500–700	~30–50

Список используемых источников

1. Википедия. Сценарный язык [электронный ресурс].

URL: https://ru.wikipedia.org/wiki/Сценарный_язык (дата обращения:

25.06.2026).

2. Википедия. Tcl [электронный ресурс].

URL: https://ru.m.wikipedia.org/wiki/Tcl (дата обращения: 25.06.2026).

3. Ousterhout, J. Scripting: Higher-Level Programming for the 21st Century / John

Ousterhout. – IEEE Computer, 1998.

4. Репозиторий GitHub. languages-benchmark [электронный ресурс].

URL: https://github.com/rodiongork/languages-benchmark (дата обращения:

25.06.2026).

5. Репозиторий GitHub. best-python-ide-for-an-old-computer [электронный

ресурс]. URL:

https://raw.githubusercontent.com/izikeros/blog/master/content/posts/notes/best_python_ide_for_an_old_computer.md (дата обращения: 25.06.2026).

6. University of Toronto. CSC401: Introducing Python [электронный ресурс].

http://www.cs.toronto.edu/~frank/csc2501/Tutorials/2501_python_web/pyintro.html (дата обращения: 25.06.2026).

7. Блог Tutortop. IDE Python 2025: PyCharm, VS Code, Spyder – сравнение и

выбор [электронный ресурс]. URL:
https://blog.tutortop.ru/chto-takoe-ide-dlya-python-vybiraem-idealnuyu-sredu-razrabotki/ (дата обращения: 25.06.2026).