



## Gateway

Что если бы создание структуры проекта было так же просто, как описать её словами? Представляем Муху RCD Garden — компонент, превращающий ASCII-дерево в живую файловую систему. От простой идеи до практичного инструмента — история о том, как небольшая "муха" решила большую проблему разработчиков, избавив их от рутинного создания каталогов и файлов.



## Essence

- **Муха (Fly)** в RCD Garden — это специализированный инструмент для генерации файловой структуры проекта из текстовых ASCII-диаграмм, который использует древовидную абстракцию для преобразования визуальных представлений в реальные каталоги и файлы.
- Основу мухи составляет **мозг (Brain)**, который преобразует ASCII-дерево в универсальную структуру, корректно обрабатывая уровни вложенности, символы структуры и комментарии, что решило ключевую проблему ранних версий с "проклятыми директориями".
- Дополнительную функциональность обеспечивают **лапки (Legs)**, добавляющие полезные файлы в созданную структуру: README.md с автоматически сгенерированной картой проекта, базовый .gitignore и заглушки для пустых файлов на основе их расширений.
- Муха использует **"варенье" (Jelly)** как источник данных — предварительно обработанное содержимое из файла table.txt, очищенное от комментариев, что соответствует поговорке "муха села на варенье — вот и всё стихотворенье".
- Архитектура Мухи вписывается в общую концепцию RCD Garden, где каждый инструмент представлен как животное с уникальной ролью в экосистеме разработки, а в будущем предполагается взаимодействие с другими животными, например, с Пауком, который будет "ловить" мух разных типов.



## Panorama



## Рождение идеи из практической необходимости

Всё началось с рутинной, но раздражающей проблемы: каждый новый проект требует создания множества каталогов и файлов вручную. Разработчики часто получают

структуру проекта в виде ASCII-диаграммы от коллеги или генерируют её с помощью AI, но затем тратят время на механическое воссоздание этой структуры в файловой системе.

```
project/
├── src/
│   ├── components/
│   │   ├── Button.js
│   │   └── Header.js
│   └── index.js
└── package.json
```

Такие диаграммы наглядны, но не функциональны: нельзя просто взять и запустить ASCII-дерево. Приходится создавать каждый каталог и файл по одному, что особенно неэффективно для больших проектов с глубокой вложенностью.

В этот момент родилась идея "Мухи" — компонента RCD Garden, который смог бы "сесть" на такую структуру и превратить её в реальную файловую систему. Название отсылает к детской поговорке "муха села на варенье — вот и всё стихотворенье", намекая на простоту концепции: муха садится на текстовое представление (варенье) и превращает его в готовый проект.

## Первые шаги и препятствия

Первая версия Мухи была примитивной: она разбивала текст по строкам и пыталась определить уровень вложенности по количеству пробелов в начале строки. Этот подход работал для простейших структур, но быстро обнаружил свои ограничения:

1. **Проблема с символами структуры:** Муха не понимала различные символы ASCII-дерева (├─, └─, │)
2. **Проблема с уровнями вложенности:** неправильно определялась глубина для сложных структур
3. **"Проклятые директории":** отдельная категория ошибок, когда комментарии в ASCII-дереве превращались в реальные каталоги

```
project/
├── src/ # Исходный код
└── components/ # UI компоненты
```

В этом примере комментарии **# Исходный код** и **# UI компоненты** создавались как часть имени директории или, что еще хуже, как отдельные "проклятые" директории, нарушая всю структуру.

Попытки использовать регулярные выражения для решения проблемы приводили к еще более запутанному коду, который все равно ломался на сложных примерах. Требовался принципиально новый подход.



## Прорыв: древовидная абстракция

Решающий момент пришел с идеей: вместо попыток парсить ASCII-дерево напрямую, нужно сначала преобразовать его в абстрактную древовидную структуру в памяти, а затем уже эту структуру преобразовывать в файлы и каталоги.

Этот подход привел к созданию целостной архитектуры Мухи:

1. **Мозг (Brain)** — основной компонент, который преобразует ASCII-дерево в универсальную структуру и затем создает файловую систему:

```
class Brain {
  // Преобразует ASCII-дерево в список путей
  parseTree(asciiTree) {
    // Строим дерево в памяти
    const tree = this.buildTree(asciiTree);

    // Преобразуем дерево в список путей
    const paths = [];
    this.treeToPathList(tree, "", paths);

    return paths;
  }

  // Создает файловую структуру на основе списка путей
  createStructure(paths, baseDir) {
    // Сначала создаем все каталоги
    // Затем создаем файлы
  }
}
```

2. **Варенье (Jelly)** — источник данных, который читает и предварительно обрабатывает содержимое из файла table.txt:

```
class Jelly {
  getData() {
    // Получаем исходное содержимое
    const rawData = fs.readFileSync(this.tablePath, 'utf8');

    // Предварительная обработка - очистка комментариев
    const cleanedData = this.cleanComments(rawData);

    return cleanedData;
  }
}
```

3. **Лапки (Legs)** — вспомогательные функции, которые добавляют полезные файлы к созданной структуре:

```
// Лапки мухи
const legs = {
  // Создает README.md с описанием структуры
  leg1: createReadmeOnTheFly,

  // Создает базовый .gitignore
  leg2: createGitignore,

  // Создает заглушки для пустых файлов
  leg3: createFileStubs
};
```

Такая модульная архитектура сделала код более организованным и расширяемым, а главное — позволила элегантно решить ключевые проблемы.

## Решение "проклятых директорий"

Особое внимание было уделено обработке комментариев в ASCII-дереве. Вместо сложных функций "лечения" уже созданных "проклятых директорий", проблема была решена на стадии построения дерева:

```
buildTree(asciiTree) {
  // Для каждой строки...
  for (let i = 0; i < lines.length; i++) {
    const line = lines[i].trimRight();

    // Пропускаем пустые строки и строки-комментарии
    if (!line.trim() || line.trim().startsWith('#')) {
      continue;
    }

    // Ищем комментарий в строке и игнорируем его часть
    const commentIndex = line.indexOf('#');
    const processLine = commentIndex !== -1
      ? line.substring(0, commentIndex).trimRight()
      : line;

    // Пропускаем, если строка стала пустой после удаления комментария
    if (!processLine.trim()) {
      continue;
    }

    // Дальнейшая обработка...
  }
}
```

```
}
```

Этот элегантный подход позволил полностью исключить "проклятые директории" из создаваемой структуры, сохраняя при этом всю полезную информацию из ASCII-дерева.



## От мухи к пауку: эволюция концепции

Успех с мухой породил идею дальнейшего развития экосистемы RCD Garden. Если муха специализируется на конкретном формате представления структуры (ASCII-дерево), то почему бы не создать других специализированных "насекомых" для других форматов?

Так родилась концепция "**Паука**" — более сложного животного, которое может "ловить" разных мух:

1. **ASCII-муха**: для работы с ASCII-деревьями (текущая реализация)
2. **Markdown-муха**: для структур, представленных в Markdown формате
3. **JSON-муха**: для работы с JSON-описаниями
4. **HTML-муха**: для HTML-представлений структуры

Паук выступал бы как координатор, выбирающий подходящую муху для обработки конкретного входного формата и создающий универсальное представление структуры.

```
class Spider {
  catchFly(content) {
    // Выбираем подходящую муху для обработки содержимого
    for (const fly of this.flies) {
      if (fly.canProcess(content)) {
        return fly;
      }
    }
    return null;
  }

  weaveWeb(filePath, outputDir) {
    // Читаем содержимое файла
    const content = fs.readFileSync(filePath, 'utf8');

    // Ловим подходящую муху
    const fly = this.catchFly(content);

    // Преобразуем содержимое в универсальную структуру
    const universalTree = fly.parseToTree(content);

    // Создаем структуру на диске
    this.createStructure(universalTree, outputDir);
  }
}
```

```
}  
}
```

Эта идея открыла путь к еще более амбициозной концепции — использованию паука как своего рода "операционной системы для каталогов", которая могла бы динамически управлять сложными структурами проектов, обеспечивая интеллектуальную маршрутизацию и визуализацию.



## Расширение экосистемы: новые животные

В процессе работы с мухой стало ясно, что RCD Garden может включать и других "животных" со специализированными ролями:

- **Крабы (Crabs)** — отвечают за тестирование. Каждый файл с тестом начинается с префикса `crab_`, а центральный координатор тестов называется `big_crab.js`.
- **Осьминог (Octopus)** — потенциальное решение для управления множественными однородными элементами (например, массовое создание и переименование файлов).

Эти идеи продолжают биомиметическую метафору RCD Garden, где каждый компонент системы представлен как животное с определенными способностями и ролью в экосистеме.



## Практическая польза и перспективы

В своем текущем состоянии Муха RCD Garden уже представляет собой полезный инструмент для разработчиков:

1. **Экономия времени:** создание структуры проекта сокращается с десятков минут до секунд
2. **Уменьшение ошибок:** автоматическое создание предотвращает опечатки и пропуски
3. **Стандартизация:** автоматически добавляемые README.md и .gitignore обеспечивают базовую документацию и настройки

Будущие расширения, такие как Паук и Осьминог, обещают еще большую продуктивность, особенно для сложных проектов с большим количеством компонентов.



**Ключевая мантра RCD Garden:** "Каждый каталог — это класс, каждый файл — метод, каждое животное — живой организм в экосистеме разработки."

Муха, как первое полностью функциональное животное в этой экосистеме, демонстрирует практическую ценность биомиметического подхода к организации кода и инструментов разработки.



## Текущее состояние и следующие шаги

На данный момент Муха RCD Garden реализована и успешно функционирует, с следующими компонентами:

- **fly.js** — главный файл, координирующий работу мухи
- **fly\_brain.js** — мозг мухи (логика парсинга)
- **fly\_jelly.js** — варенье для мухи (источник данных)
- **fly\_legs.js** — лапки мухи (вспомогательные функции)
- **table.txt** — файл для вставки ASCII-структуры

Ближайшие шаги по развитию включают:

1. Реализацию Паука с поддержкой разных типов мух
2. Расширение функциональности лапок (например, добавление шаблонов для разных типов файлов)
3. Создание простого CLI-интерфейса для удобного использования
4. Интеграцию с существующими инструментами разработки

В более долгосрочной перспективе планируется разработка полноценного графического интерфейса для визуализации и управления структурами проектов, что приблизит систему к концепции "операционной системы для каталогов".

Муха, начавшись как простое решение конкретной проблемы, стала первым шагом к созданию целостной экосистемы инструментов разработки, объединенных общей биомиметической метафорой и философией.