DOI 10.37882/2223-2966.2025.06.15

ЭРГОНОМИЧЕСКИЙ АНАЛИЗ МЕНЕДЖЕРОВ ПАКЕТОВ В ВЕБ-РАЗРАБОТКЕ

ERGONOMIC ANALYSIS OF PACKAGE MANAGERS IN WEB DEVELOPMENT

B. Goryachkin
P. Strikhar
I. Bondarenko
V. Khizhnyakov

Summary. *Problem statement*. In the modern world of programming, especially in the context of web development, JavaScript has become one of the most popular and widely used languages. Package managers are tools that automate the process of installing, updating, and removing libraries and modules necessary for applications to work. Today, there are several popular package managers for JavaScript, including npm, yarn, and pnpm. Each of them has its own characteristics, advantages, and disadvantages, which will be discussed in this article.
Goal. Analysis and research of various package managers for JavaScript, such as npm, yarn, and pnpm, with an emphasis on their functionality, performance, and usability. The research aims to identify the advantages and disadvantages of each of the tools, as well as to determine their impact on the effectiveness of web application development.
Results. As a result, it is planned to get a comprehensive understanding of modern package managers for JavaScript, their functionality and impact on the development process. It is expected that the results of the study will help developers make a more informed choice of tools for dependency management, as well as contribute to the further development of web development practices.
Practical significance. Studying the architecture of the above-mentioned tools, measuring the execution time of operations and the use of system resources will determine the difference between package managers. Choosing the most appropriate package manager will significantly reduce resources (especially time) when executing commands that are executed by interface developers daily.

Keywords: Package managers, npm, pnpm, yarn, architecture, performance comparison, metrics, dependency management, JavaScript, software development, build optimization.

Горячкин Борис Сергеевич
кандидат технических наук, доцент,
Московский государственный технический университет им. Н.Э. Баумана
bsgor@mail.ru
Стрихар Павел Андреевич
Московский государственный технический университет им. Н.Э. Баумана
p.strikhar@gmail.com
Бондаренко Иван Геннадьевич
Московский государственный технический университет им. Н.Э. Баумана
ivan.frinom@gmail.com
Хижняков Вадим Максимович
Московский государственный технический университет им. Н.Э. Баумана
vadimkhiz@mail.ru

Аннотация. *Постановка проблемы*. В современном мире программирования, особенно в контексте веб-разработки, язык JavaScript стал одним из наиболее популярных и широко используемых. Пакетные менеджеры являются инструментами, которые автоматизируют процесс установки, обновления и удаления библиотек и модулей, необходимых для работы приложений. На сегодняшний день существует несколько популярных пакетных менеджеров для JavaScript, среди которых npm, yarn и pnpm. Каждый из них имеет свои особенности, преимущества и недостатки, которые будут рассмотрены в данной статье.
Цель. Анализ и исследование различных пакетных менеджеров для JavaScript, таких как npm, yarn и pnpm, с акцентом на их функциональность, производительность и удобство использования. Исследование направлено на выявление преимуществ и недостатков каждого из инструментов, а также на определение их влияния на эффективность разработки веб-приложений
Результаты. В результате выполнения планируется получить комплексное представление о современных пакетных менеджерах для JavaScript, их функциональности и влиянии на процесс разработки. Ожидается, что результаты исследования помогут разработчикам сделать более обоснованный выбор инструментов для управления зависимостями, а также внесут вклад в дальнейшее развитие практик веб-разработки.
Практическая значимость. Изучение архитектуры вышеупомянутых инструментов, измерение времени выполнения операций и использования системных ресурсов позволит определить разницу между пакетными менеджерами. Выбор наиболее подходящего менеджера пакетов позволит заметно сократить ресурсы (особенно временные) при исполнении команд, которые выполняются разработчиками интерфейсов на ежедневной основе.

Ключевые слова: пакетные менеджеры, npm, pnpm, yarn, архитектура, сравнение производительности, метрики, управление зависимостями, JavaScript, разработка программного обеспечения, оптимизация сборки.

Введение

JavaScript стал ключевым языком в веб-разработке, что привело к необходимости эффективного управления зависимостями. Пакетные менеджеры автоматизируют процесс установки, обновления и удаления библиотек и модулей, позволяя разработчикам сосредоточиться на написании кода. В крупных проектах может использоваться множество пакетов с различными зависимостями, что делает выбор правильного пакетного менеджера особенно важным.

Среди наиболее популярных пакетных менеджеров для JavaScript выделяются npm, yarn и pnpm. Каждый из них имеет свои особенности и преимущества, что требует от разработчиков понимания их различий для выбора оптимального инструмента. Менеджеры пакетов играют ключевую роль в сборке проектов, обеспечивая автоматическую установку, обновление и управление зависимостями, что упрощает процесс разработки и гарантирует совместимость библиотек [1]. Эргономика этих инструментов также влияет на продуктивность программистов, поскольку удобство использования может ускорить процесс разработки и повысить качество кода [2].

Основные функции пакетных менеджеров

Пакетные менеджеры играют ключевую роль в разработке программного обеспечения, позволяя разработчикам быстро устанавливать необходимые библиотеки и инструменты с помощью простых команд. Это значительно ускоряет процесс разработки, устраняя необходимость вручную загружать и настраивать каждую библиотеку. Одним из основных преимуществ пакетных менеджеров является автоматическое отслеживание зависимостей, что упрощает работу разработчиков [3].

В веб-разработке наиболее популярными пакетными менеджерами являются npm, Yarn и pnpm. Каждый из них предлагает свои преимущества: npm предоставляет доступ к обширной библиотеке пакетов, Yarn улучшает скорость установки и управление зависимостями, а pnpm экономит место на диске и повышает производительность установки.

Таким образом, пакетные менеджеры становятся неотъемлемой частью разработки, особенно в веб-сфере. Они помогают эффективно управлять зависимостями, ускоряют процесс разработки и минимизируют проблемы с версиями библиотек [4]. В дальнейшем исследовании будет рассмотрено влияние различных пакетных менеджеров на продуктивность разработки и опыт использования с точки зрения эргономики.

На схеме работы пакетного менеджера (рис. 1) разработчик начинает процесс, передавая файл package.json в пакетный менеджер, который содержит информацию о проекте, включая зависимости и версии пакетов. Пакетный менеджер анализирует этот файл, разрешает зависимости и загружает необходимые модули в локальную директорию проекта, создавая папку node_modules.

После завершения установки разработчик получает доступ к директории node_modules и передает ее сбор-

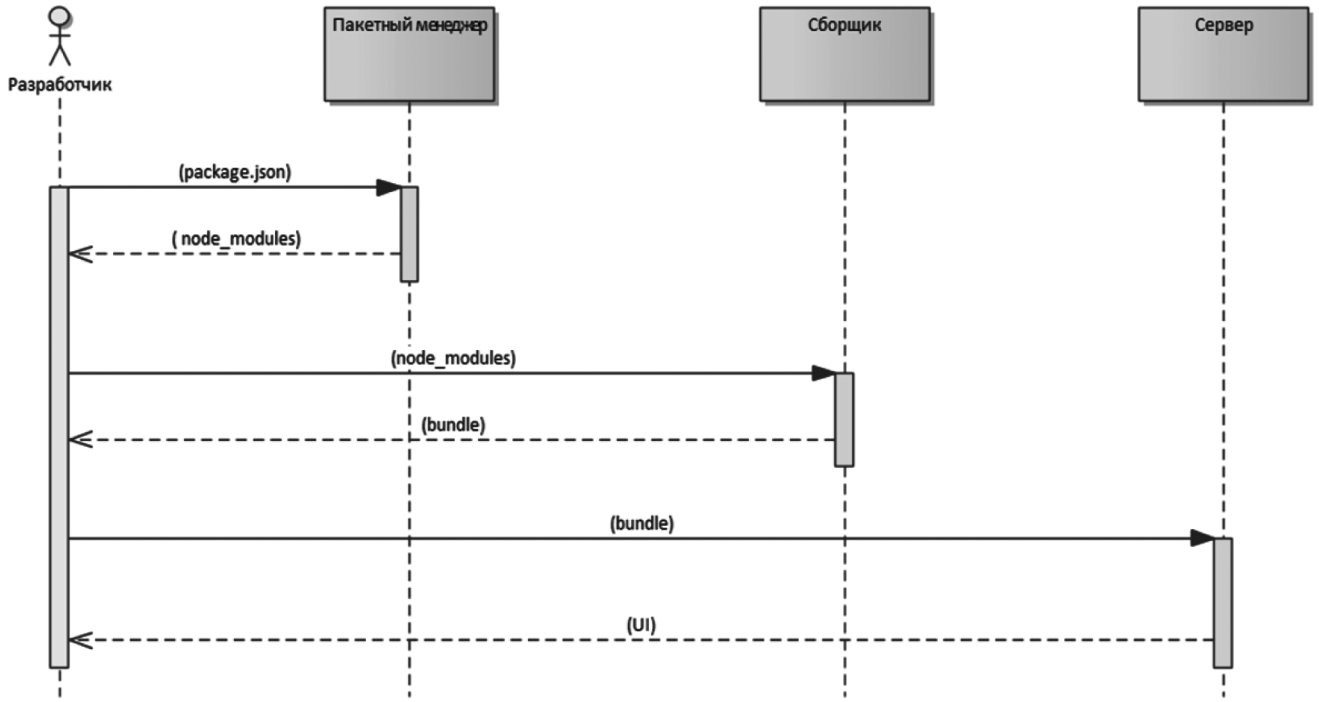

Рис. 1. Диаграмма последовательностей процесса

щику кода. Сборщик компилирует и объединяет исходный код с зависимостями, выполняя такие действия, как транспиляция и минификация, в результате чего формируется скомпилированный файл или набор файлов, называемый bundle.

Этот bundle затем загружается на сервер, где приложение становится доступным пользователям. Сервер обрабатывает запросы и возвращает UI-страницу, которая отображает интерфейс приложения и готова для взаимодействия с конечными пользователями.

Архитектурные особенности npm, yarn и pnpm

NPM (Node Package Manager) — это официальный менеджер пакетов для Node.js, который упрощает интеграцию библиотек в проекты, используя централизованный реестр для управления зависимостями [5]. Появление NPM положило конец ручной установке зависимостей, однако с выходом Yarn в 2016 году, который предложил улучшенные функции, разработчики NPM были вынуждены улучшать свой продукт. В пятой версии NPM была внедрена функция кэширования, что значительно повысило надежность установки пакетов.

Yarn, разработанный Facebook, поддерживает рабочие пространства, позволяя устанавливать зависимости для нескольких проектов одновременно, что актуально для современных практик разработки, таких как монорепозитории. В 2016 году появился PNPM, который решает проблемы объема дискового пространства и доступности зависимостей. Он выполняет операции с пакетами атомарно и использует технологию «hard links», что уменьшает объем занимаемого пространства и ускоряет установку.

Процесс установки пакетов начинается с разрешения зависимостей, где пакетный менеджер анализирует зависимости и подбирает актуальные версии библиотек. Этот процесс становится рекурсивным, поскольку у загружаемых библиотек могут быть свои зависимости. Для экономии дискового пространства введено поле devDependencies, которое содержит зависимости для разработки, устанавливаемые только если они являются прямыми зависимостями проекта. Транзитивные devDependencies пакетный менеджер игнорирует (зависимости зависимостей проекта называются транзитивными, на рисунке 2 будет проигнорирована «Библиотека 5»).

NPM имел «nested» модель установки, которая подразумевает, что для каждой зависимости проекта создаётся своя директория node_modules, в которой изолированно хранятся её зависимости — это позволяет избежать конфликтов версий. Переход на «nested» модель установки привел к образованию глубокой иерархии в директории node_modules, что занимало много места на диске и вызывало проблемы с ограничением длины путей на Windows. Хотя для бекенда это может быть приемлемо, передача такого количества библиотек на веб-сайт может быть слишком затратной, особенно из-за возможных дубликатов.

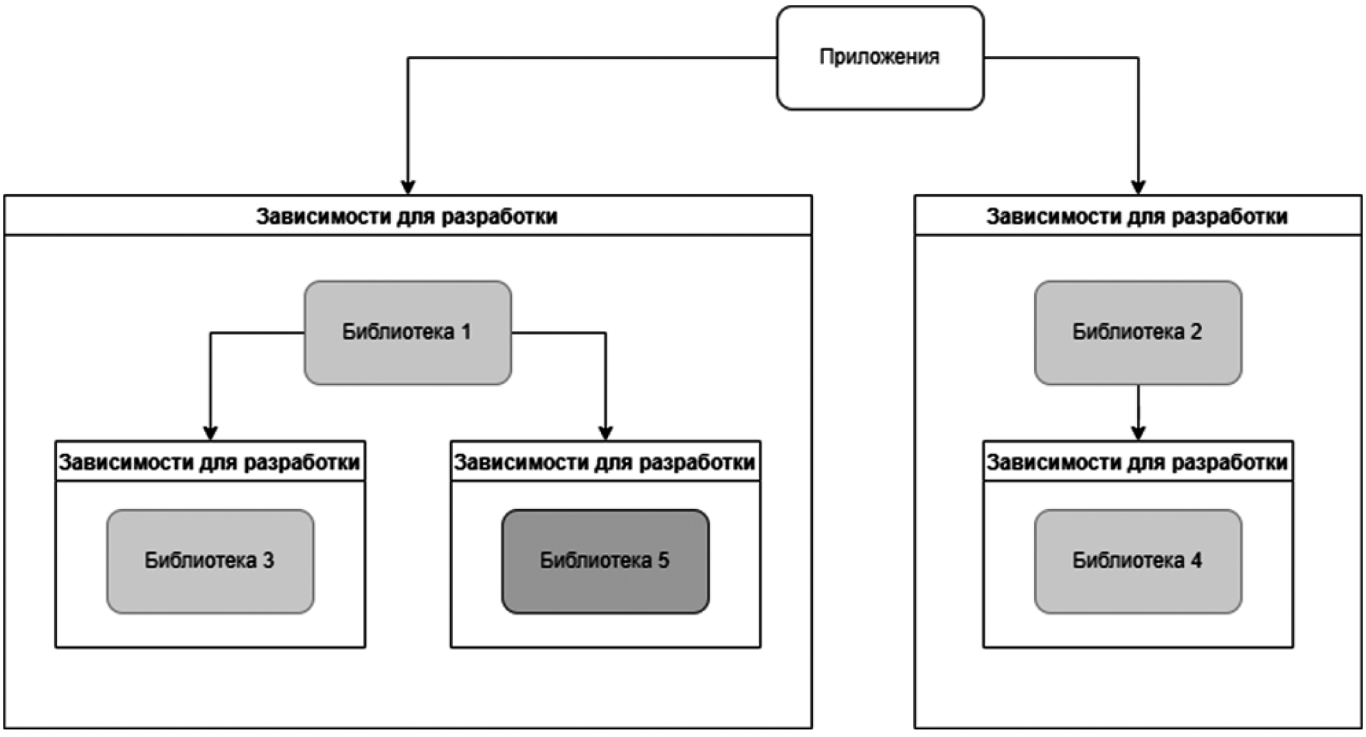

Рис. 2. Установка зависимостей с devDependencies

В связи с этим в NPM 3 была введена новая «hoisted» модель установки и механизм дедупликации пакетов. Эта модель сочетает в себе элементы плоской и «nested» моделей, позволяя хранить пакеты в верхней директории node_modules, а вложенности возникают только при конфликтах версий.

Работа этой модели обеспечивается механизмом разрешения модулей в Node.js, суть которого заключается в том, что при поиске пакета, указанного в require, Node. js проходит по всем директориям node_modules снизу вверх, то есть «всплывает» (аналогично всплытию переменных в Javascript).

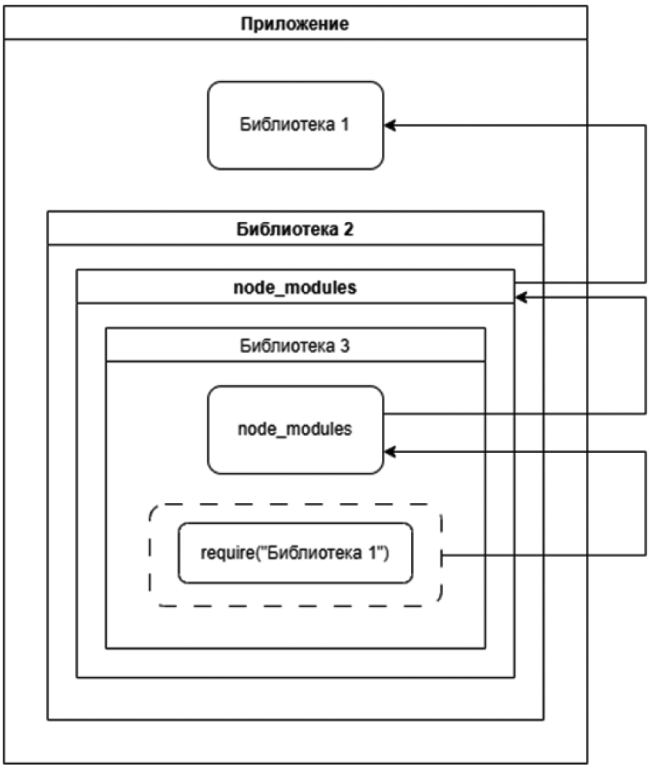

Рис. 3. Разрешение модулей в Node.js

Пакетный менеджер начинает установку с разрешения зависимостей, которые часто задаются диапазонами версий. Это создает неопределенность, так как две последовательные установки могут давать разные результаты. В процессе разработки разработчики могут столкнуться с тем, что в системе непрерывной интеграции (CI) пакетный менеджер обнаруживает возможность установки более свежей версии транзитивной зависимости, что может привести к неожиданным изменениям в поведении кода.

Для решения этой проблемы был создан альтернативный пакетный менеджер Yarn. Он генерирует файл локфайла (yarn. lock), который фиксирует конкретные версии пакетов, выбранные при установке. При следующей установке Yarn проверяет соответствие между package. json и yarn. lock, пропуская этап разрешения зависимостей и загружая пакеты из заранее определенного списка [6]. Это ускоряет процесс установки и обеспечивает предсказуемость результатов, так что две последовательные установки дают идентичный результат, даже на разных машинах. Этот подход позже был реализован и в npm.

Помимо вышеописанного механизма фиксации версий зависимостей Yarn также имел ряд других преимуществ перед NPM. Основная причина быстроты Yarn — кэш. Он позволяет создать на своей машине собственный реестр пакетов, чтобы в процессе установки заменять сетевой запрос на копирование папок в файловой системе. Меньше сетевых запросов — меньше времени занимает установка.

PNPM в отличие от NPM и Yarn не пытается сделать структуру node_modules как можно более плоской, вместо этого он скорее нормализует граф зависимостей. После установки PNPM создаёт в node_modules директорию. pnpm, которая концептуально представляет собой хранилище ключ-значение, в котором ключом является название пакета и его версия, а значением — содержимое этой версии пакета [7]. Такая структура данных

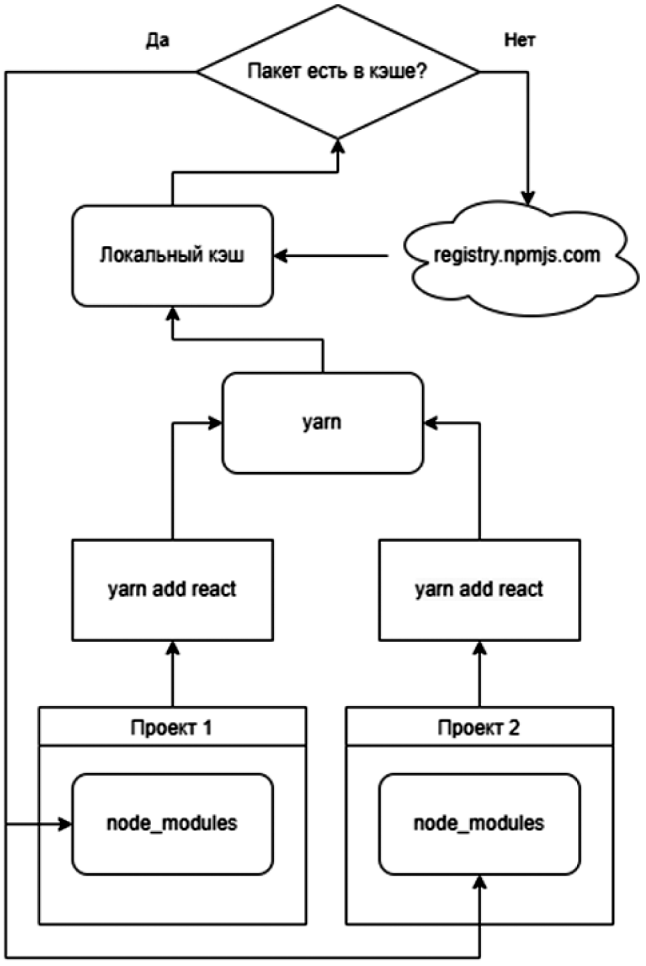

Рис. 4. Кэш пакетного менеджера

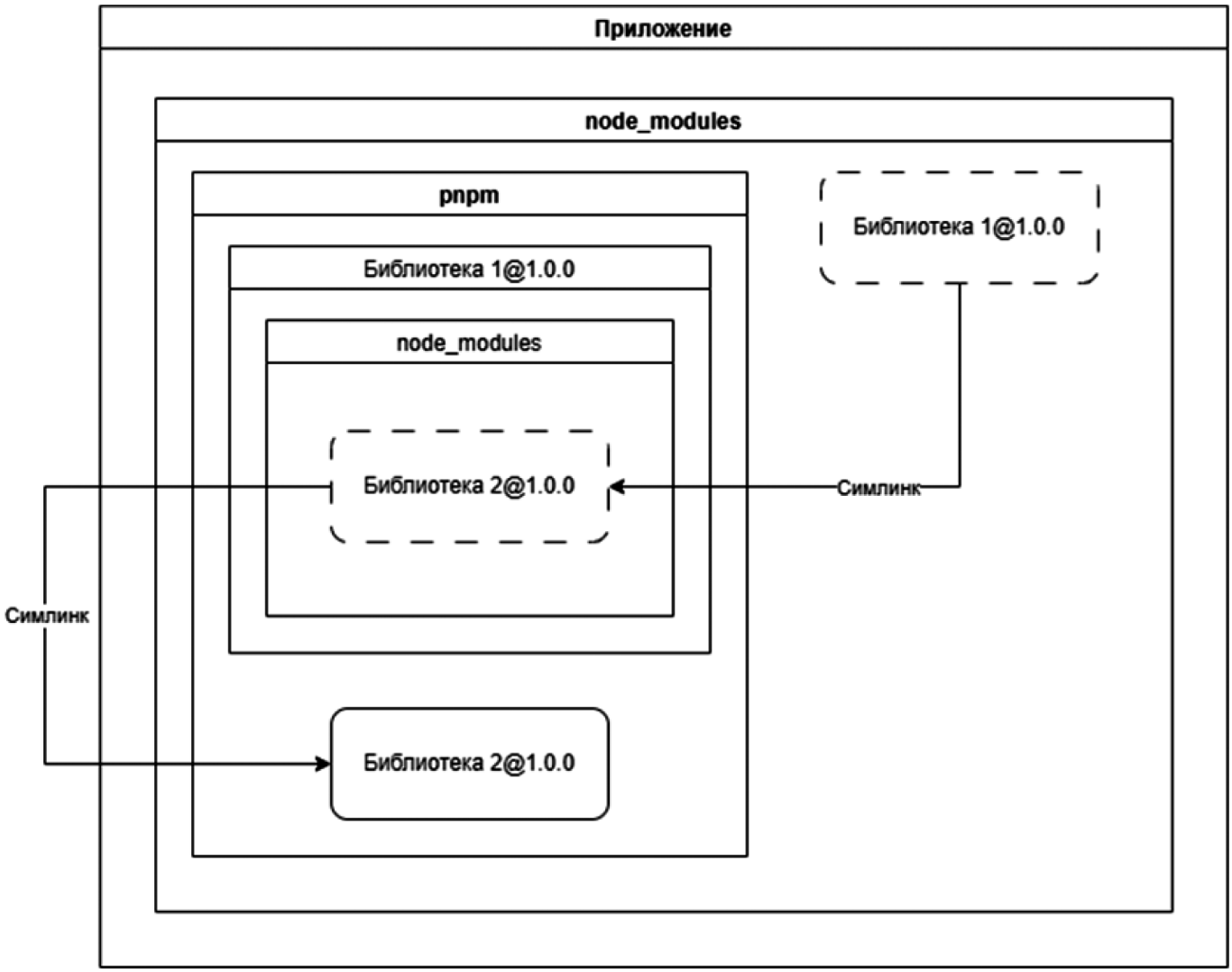

Рис. 5. Структура node_modules с PNPM

исключает возможность возникновения дубликатов. Структура самой директории node_modules будет подобна «nested»-модели из NPM, но вместо физических файлов ней будут находиться симлинки, которые ведут в то самое хранилище пакетов.

В node_modules каждого пакета будут находиться только симлинки на те пакеты, которые указаны у него в package.json. PNPM может создать директорию. pnpm не только в node_modules проекта, но и глобально. В таком случае node_modules у проектов будут содержать только симлинки, за счёт чего ускоряется установка зависимостей (создание симлинка занимает меньше времени, чем копирование файлов) и экономится количество дискового пространства.

Эргономические критерии для оценки пакетных менеджеров

Первый эргономический признак — время первичной установки пакетов. Это один из самых критичных факторов, так как разработчики часто сталкиваются с необходимостью настраивать окружение для новых проектов или обновлять существующие.

$$T_{npminstall} = T_{download} + T_{install} + T_{setup} \quad (1)$$

$$T_{yarninstall} = T_{download} + T_{install} + T_{setup} \quad (2)$$

$$T_{pnpminstall} = T_{download} + T_{install} + T_{setup} + T_{linking} \quad (3)$$

Где: $T_{download}$ — время, затраченное на загрузку пакетов из реестра.

$T_{install}$ — время, затраченное на установку загруженных пакетов в проект.

T_{setup} — время, необходимое для выполнения начальной конфигурации (например, создание структуры папок).

$T_{linking}$ — время, затраченное на создание жестких ссылок для общих зависимостей.

Npm и yarn имеют схожие подходы к скачиванию и установке пакетов. Однако, yarn использует кэширование, что может ускорить процесс. Pnpm использует уникальную стратегию хранения пакетов, что может добавить время на связывание, но в целом его подход к кэшированию может значительно ускорить повторные установки.

Следующий признак — время переустановки пакетов. В процессе работы над проектом могут возникать ситуации, когда необходимо переустановить определенные пакеты, например, после обновления или при возникновении конфликтов.

$$T_{npmreinstall} = T_{download} + T_{install} \quad (4)$$

$$T_{yarnreinstall} = T_{download} + T_{install} \quad (5)$$

$$T_{pnpmreinstall} = T_{linking} \quad (6)$$

Где: $T_{download}$ — время повторной загрузки пакетов.

$T_{install}$ — время повторной установки пакетов.

$T_{linking}$ — время, затраченное на создание жестких ссылок для общих зависимостей.

Для переустановки пакетов, npm и yarn будут тратить время на повторное скачивание, в то время как pnpm может использовать существующие пакеты в кэше, что делает его более эффективным. Поскольку pnpm использует кэшированные версии пакетов, время переустановки включает только связывание.

Время установки дополнительных пакетов — это еще один важный аспект. Веб-разработка требует гибкости и быстрого реагирования на изменения требований проекта.

$$T_{npmadd} = T_{download} + T_{install} \quad (7)$$

$$T_{yarnadd} = T_{download} + T_{install} \quad (8)$$

$$T_{pnpmadd} = T_{linking} \quad (9)$$

Где: $T_{download}$ — время загрузки пакетов.

$T_{install}$ — время установки пакетов.

$T_{linking}$ — время, затраченное на создание жестких ссылок для общих зависимостей.

Время обновления версий пакетов является критичным для поддержания актуальности проекта.

$$T_{npmupdate} = T_{checkversions} + T_{downloadnewversions} + T_{installnewversions} \quad (10)$$

$$T_{yarnupdate} = T_{checkversions} + T_{downloadnewversions} + T_{installnewversions} \quad (11)$$

$$T_{pnpmupdate} = T_{checkversions} + T_{linkingnewversions} \quad (12)$$

Где: $T_{checkversions}$ — время, затраченное на проверку доступных обновлений.

$T_{downloadnewversions}$ — время, необходимое для загрузки новых версий.

$T_{installnewversions}$ — время на установку новых версий.

$T_{linkingnewversions}$ — время связывания обновляемых пакетов.

Обновление пакетов может занять больше времени в npm и yarn из-за необходимости загрузки новых версий, тогда как pnpm может оптимизировать этот процесс за счет использования ссылок на существующие пакеты.

Конечный размер дерева зависимостей также является значимым аспектом. Чем меньше размер дерева зависимостей, тем проще управлять проектом и меньше вероятность возникновения конфликтов между пакетами.

$$S = S_{npm} = S_{yarn} = |D| \quad (13)$$

$$S_{pnpm} \ll S \quad (14)$$

Где: $|D|$ — общее количество зависимостей

Pnpm использует жесткие ссылки на общие зависимости, что позволяет значительно уменьшить размер дерева зависимостей по сравнению с npm и yarn.

Наконец, конечный размер установленных пакетов на диске влияет на производительность системы в целом.

$$M = M_{npm} = M_{yarn} = \sum_{i=1}^{N} |P_i| \quad (15)$$

$$M_{pnpm} < M \quad (16)$$

Где: N — общее количество пакетов.

$|P_i|$ — размер каждого установленного пакета. У pnpm размер меньше благодаря общему хранилищу зависимостей.

За счет использования общего хранилища для зависимостей pnpm позволяет существенно сократить занимаемое пространство на диске по сравнению с npm и yarn. Все перечисленные эргономические признаки имеют прямое влияние на продуктивность разработки и общий опыт использования пакетных менеджеров.

Результаты вычислительного эксперимента

Запуски производились на одной и той же машине, чтобы нивелировать различия в производительности устройств пользователей. Технические характеристики машины, на которой проводились замеры:

— Наименование — Apple MacBook Pro, M1 Pro, 2021
— ОЗУ — 32 Гб
— ОС — MacOS Sonoma 14.2.1

После настройки стендов были проведены замеры различных метрик (табл. 1). Каждая метрика была измерена для каждого из пакетных менеджеров (npm, yarn и pnpm). Для минимизации погрешности идентичные замеры были проведены трижды и было взято среднее значение (табл. 2).

Таблица 1.
Экспериментальные замеры эргономических признаков

Эргономические признаки	Итерация	npm	yarn	pnpm
Время первичной установки пакетов, с	Итерация 1	49,602	47,456	18,747
	Итерация 2	42,431	41,873	16,747
	Итерация 3	56,221	48,794	17,899
Время переустановки пакетов, с	Итерация 1	1,251	0,494	0,602
	Итерация 2	1,193	0,297	0,613
	Итерация 3	1,214	0,267	0,556
Время установки дополнительных пакетов, с	Итерация 1	1,235	0,842	1,379
	Итерация 2	1,152	0,955	1,516
	Итерация 3	1,193	0,947	1,378
Время удаления пакетов, с	Итерация 1	1,177	0,878	1.203
	Итерация 2	1,117	0,829	1,143
	Итерация 3	1,183	0,826	1,048
Время обновления версий пакетов, с	Итерация 1	1,231	0,742	1,238
	Итерация 2	1,153	0,957	1,318
	Итерация 3	1,109	0,841	1,257
Время запуска скриптов, с	Итерация 1	0,116	0,161	0,153
	Итерация 2	0,124	0,128	0,156
	Итерация 3	0,124	0,159	0,158
Конечный размер дерева зависимостей, строк	Итерация 1	16146	7751	9879
	Итерация 2	16146	7751	9879
	Итерация 3	16146	7751	9879
Конечный размер установленных пакетов на диске, Мб	Итерация 1	301,617152	356,446	288,322
	Итерация 2	301,617152	356,446	288,322
	Итерация 3	301,617152	356,446	288,322

Pnpm показал наилучший результат по времени первичной установки пакетов, что делает его предпочтительным выбором для проектов с большим количеством зависимостей или для разработчиков, которые часто создают новые проекты.

Таблица 2.
Итоговая таблица с замерами эргономических признаков

Эргономические признаки	npm	yarn	pnpm
Время первичной установки пакетов, с	49,418	46,041	17,798
Время переустановки пакетов, с	1,219	0,353	0,59
Время установки дополнительных пакетов, с	1,193	0,915	1,424
Время удаления пакетов, с	1,159	0,844	1,096
Время обновления версий пакетов, с	1,164	0,847	1,271
Время запуска скриптов, с	0,121	0,149	0,156
Конечный размер дерева зависимостей, строк	16146	7751	9879
Конечный размер установленных пакетов на диске, Мб	301,617	356,446	288,322

Yarn оптимизирует структуру зависимостей, уменьшая количество повторяющихся зависимостей. Pnpm также направлен на оптимизацию структуры зависимостей, но его подход отличается от yarn — он использует глобальный кэш и жесткие ссылки. Yarn показывает лучшие результаты и по другим признакам, таким как время запуска скриптов, время обновления версий пакетов и пр. (см. рисунок 9), однако необходимо упомянуть, что в этих операциях, как правило, разница не превышает одной секунды.

Результаты показали, что pnpm продемонстрировал наилучшие показатели по времени первичной установки пакетов. Это делает его предпочтительным выбором для проектов с большим количеством зависимостей. Преимущества pnpm заключаются в использовании глобального кэша и создании жестких ссылок на зависимости, что значительно ускоряет процесс установки и экономит место на диске. Кроме того, pnpm устанавливает зависимости параллельно и эффективно управляет версиями пакетов. Yarn же уменьшает количество дублирующихся зависимостей за счет использования файла yarn.lock, который фиксирует версии библиотек. Алгоритм разрешения зависимостей в yarn помогает избежать установки нескольких версий одной и той же библиотеки.

Заключение

Таким образом, мы пришли к важным выводам о значимости выбора инструментального обеспечения для

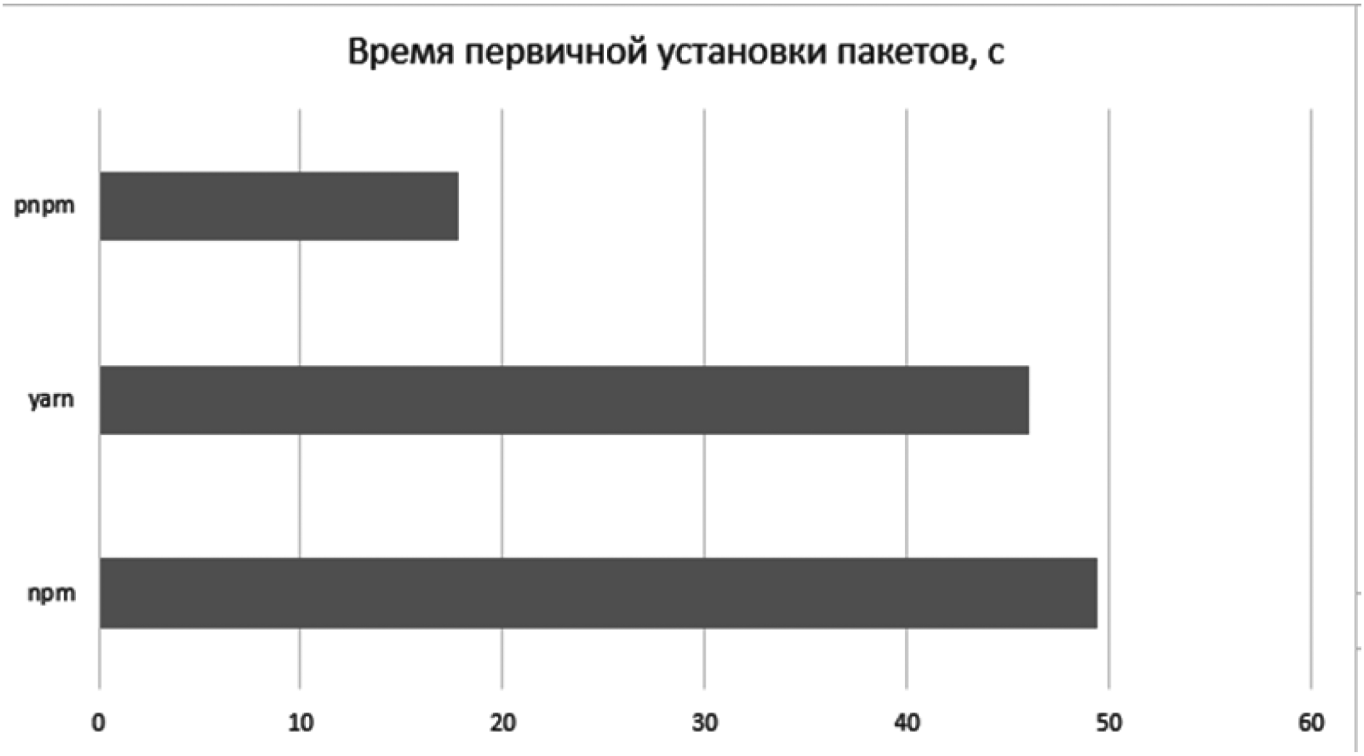

Рис. 6. Сравнение времени первичной установки пакетов

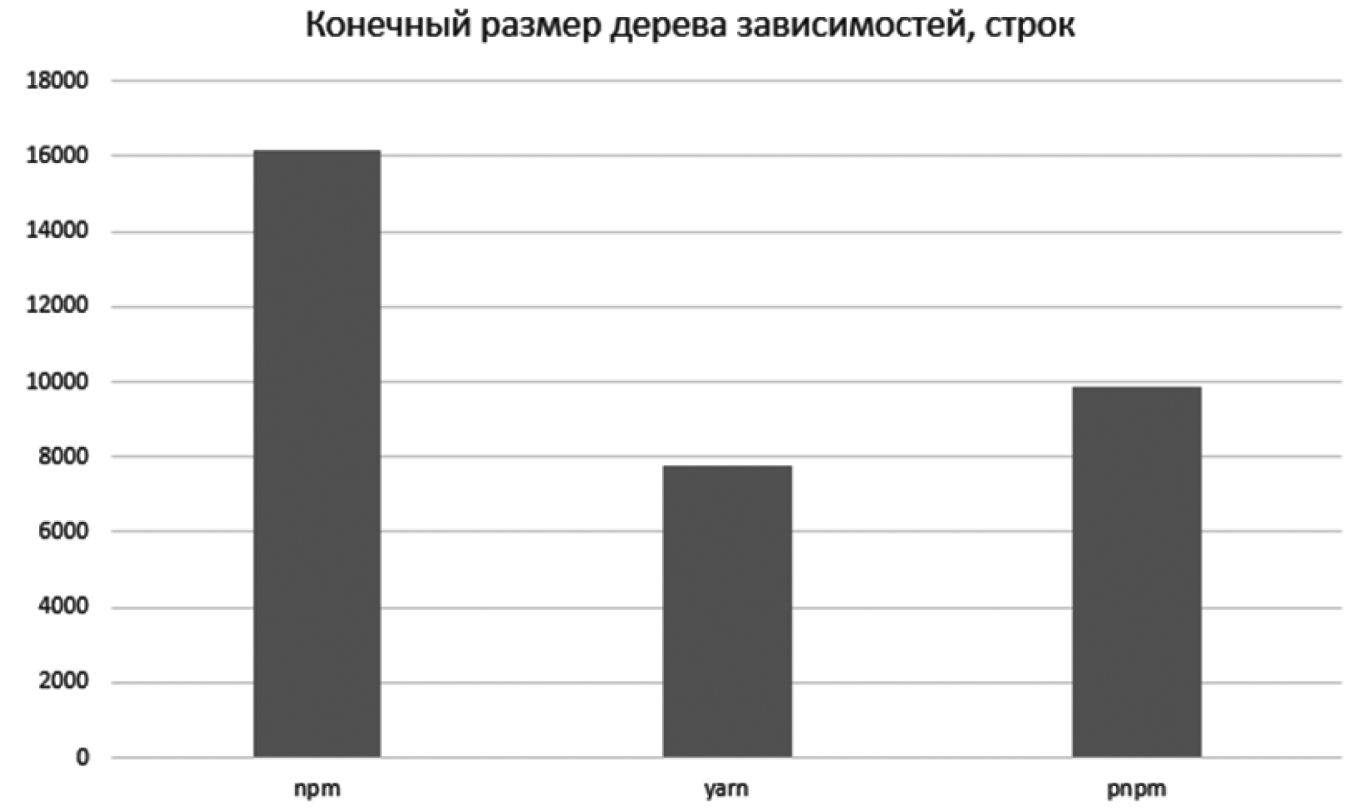

Рис. 7. Сравнение конечного размера установленных пакетов на диске

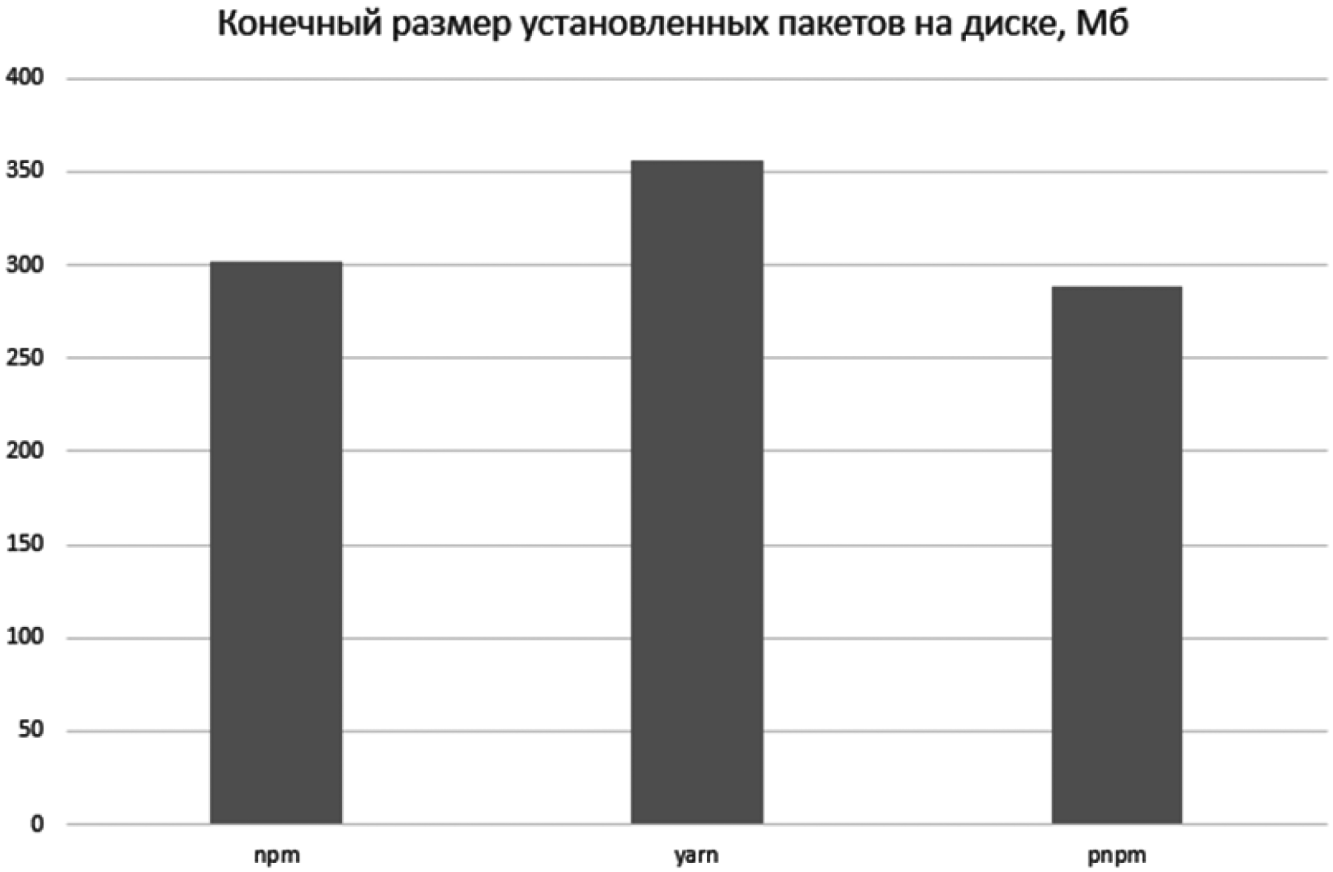

Рис. 8. Сравнение конечного размера дерева зависимостей

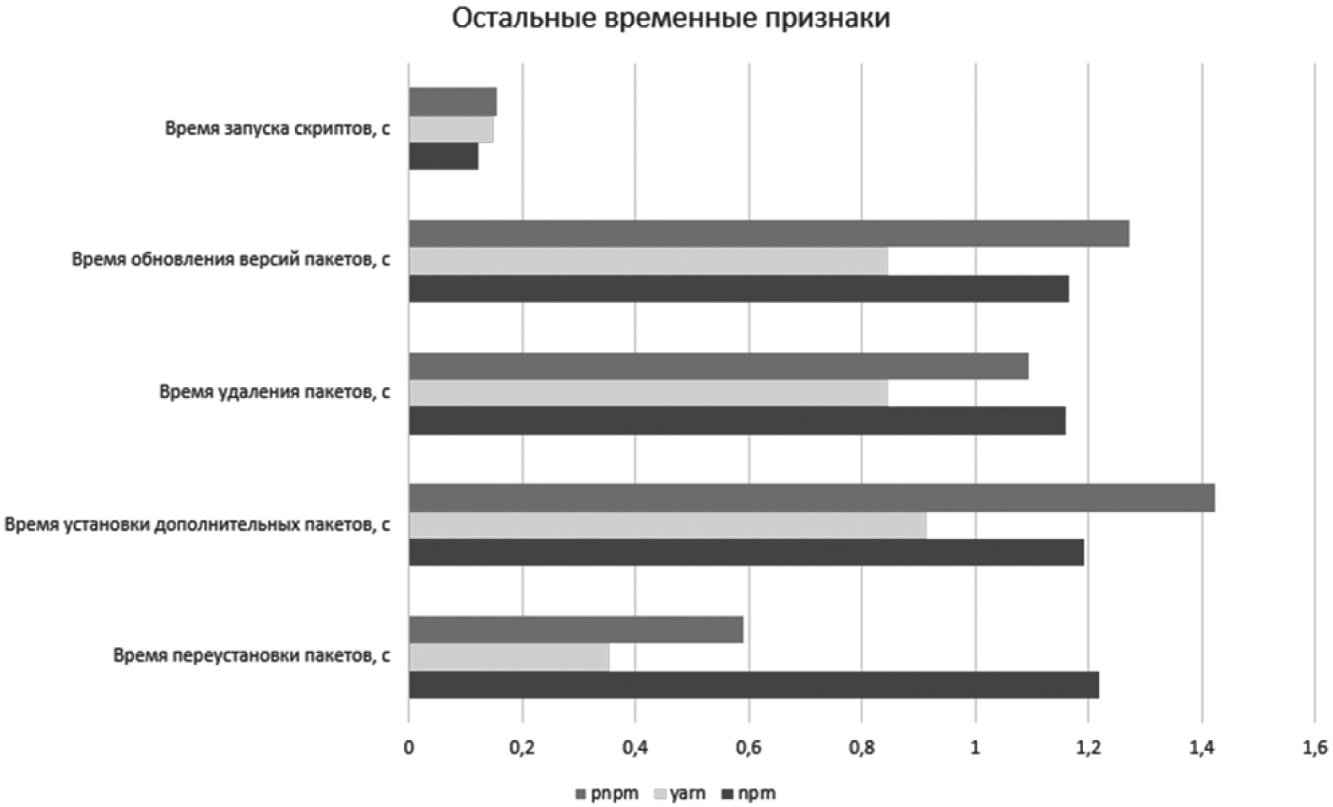

Рис. 9. Сравнение остальных временных признаков

управления зависимостями в веб-проектах. Рассмотрев основные характеристики пакетных менеджеров, мы отметили, что npm, будучи первым и наиболее распространенным, имеет недостатки, такие как медленная установка и проблемы с дублированием пакетов, что снижает его привлекательность для оптимизации рабочего процесса.

Yarn, появившийся как ответ на ограничения npm, предлагает более быструю установку и предсказуемое управление зависимостями благодаря lock-файлам. Однако он также имеет недостатки, включая сложности в управлении монорепозиториями. В отличие от них, pnpm выделяется своей эргономичностью: его подход к установке зависимостей с использованием жестких ссылок и кэширования экономит дисковое пространство и время.

Несмотря на преимущества pnpm, yarn остается серьезным конкурентом с множеством полезных функций. Правильный выбор пакетного менеджера может существенно повлиять на продуктивность разработки и качество конечного продукта.

ЛИТЕРАТУРА

1. Горячкин Б.С. Эргономический анализ систем обработки информации и управления // Вестник евразийской науки. 2017. Т. 9. №3.
2. Горячкин Б.С., Ханмурзин Т.И. Повышение эффективности работы с веб-ресурсом за счет инструментария системного программиста // Динамика сложных систем — XXI век. 2022. Т. 16, № 3.
3. Diomidis Spinellis. Package Management Systems // IEEE Software (Volume: 29, Issue: 2, March-April 2012)
4. Package management basics [Электронный ресурс] — https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Client-side_tools/Package_management?clckid=47cb7e23 (дата обращения 13.12.2024)
5. Документация npm [Электронный ресурс] — https://docs.npmjs.com/ (дата обращения 13.12.2024)
6. Документация yarn [Электронный ресурс] — https://classic.yarnpkg.com/en/docs (дата обращения 13.12.2024)
7. Документация pnpm [Электронный ресурс] — https://pnpm.io/motivation (дата обращения 13.12.2024)