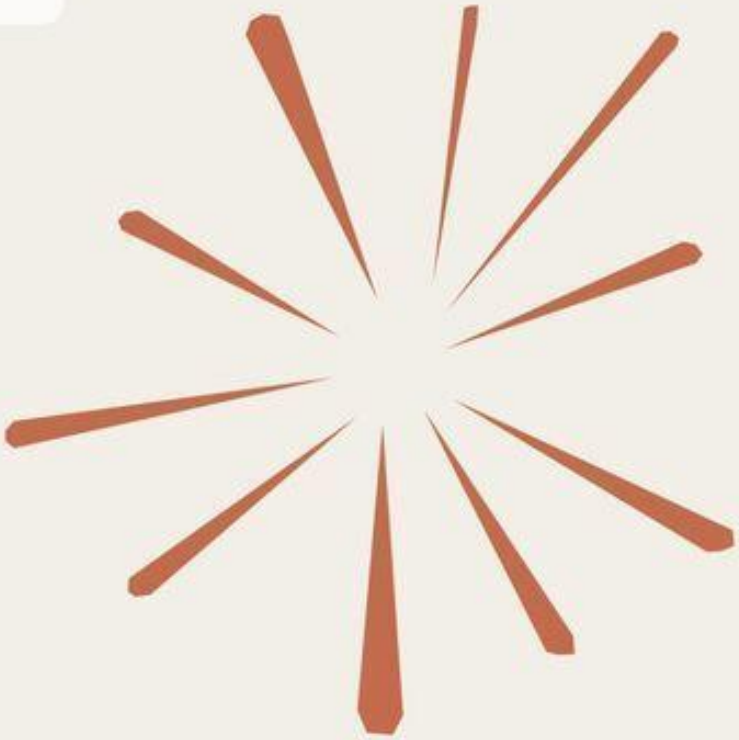
12+



AI Дизайн

AI - дизайнер на скорости света

Станислав Молчанов

Станислав Молчанов

AI Дизайн. AI-дизайнер на скорости света

«Издательские решения»

Молчанов С.

AI Дизайн. AI-дизайнер на скорости света / С. Молчанов — «Издательские решения»,

«AI Дизайн» — гайд для UX/UI-лидеров, превращающих ИИ в со-дизайнера. Автор делится методами, которые помогут:— Внедрить Brand State Architecture и DESIGN. md, чтобы ИИ помнил бренд;— Освоить промптинг как документацию и мультимодальный ввод;— Наладить хэндофф в код через Claude Code;— Собрать AI-стек под свою роль и защитить решения данными.Эта книга является манифестом для тех, кто готов заменить рутинное исполнение на стратегическое управление AI-агентами.

Содержание

AI Дизайн
AI-дизайнер на скорости света

Станислав Молчанов

ISBN 978-5-0071-1047-1
Создано в интеллектуальной издательской системе Ridero

Введение

Привет! Если ты взял эту книгу, скорее всего, ты уже давно чувствуешь что-то, чему трудно дать точное название. Не тревогу и не страх, а скорее тихое беспокойство человека, который привык опираться на собственный опыт и вдруг замечает, что почва под ногами стала другой. Коллега на ревью вскользь упоминает, что «быстро проверил идею в Claude», и показывает то, на что у тебя ушло бы полдня. Портфолио кандидата на джуниорскую позицию выглядит как работа мидла. Оффер приходит с цифрой выше ожидаемой. Ты смотришь на всё это и понимаешь: вопрос уже не в том, меняется ли что-то. Вопрос в том, как остаться собой внутри этих перемен.

Книга, которую ты держишь в руках, написана не для тех, кто только начинает. Здесь нет объяснений, что такое UX, зачем нужна дизайн-система и как работает сетка. Я предполагаю, что ты уже знаешь, почему кнопка стоит именно здесь, а не на пять пикселей левее. Я пишу для тех, кто прожил в профессии достаточно, чтобы выработать собственный стиль мышления — и теперь стоит перед вопросом, как встроить в этот стиль мышление новое, не разрушив первое.

Я был достаточно самонадеян, чтобы думать, что накопленный опыт сам по себе является защитой. Я ошибался. Но именно то, как я обнаружил свою ошибку и что сделал с этим обнаружением, легло в основу этой книги. Мы пойдём глубже: разберём, как Claude Design и GPT-4o Vision меняют саму механику того, как дизайнер думает о задаче. Как строить системы, которые превращают одного специалиста в команду. Как не потерять профессиональную идентичность в погоне за скоростью. Где граница, за которой делегирование AI становится не инструментом, а костылём.

Если ты дочитаешь до конца, у тебя на руках окажется не набор приёмов. Окажется другой способ работать — тот, в котором твои годы в профессии не обесцениваются, а становятся именно тем, чего AI воспроизвести не может.

С этого и начнём.

Кто я такой

С 2012 по 2017 годы я был студентом и много времени проводил в играх. Параллельно начал увлекаться графическим дизайном, а потом решил перейти к веб-дизайну, потому что увидел, как многое зависит от удобного и понятного интерфейса.

В 2016—2017 годах, еще во время учёбы, я попробовал себя во фрилансе на Upwork. Брался за разные проекты — от простых баннеров до сложных концепций, и постепенно дорос до уровня «Эксперт».

С 2012 по 2019 фриланс стал моей основной деятельностью. Я получил статус Top Rated Plus, завоевал награды Awwwards и FWA, а ещё отработал свыше 5000 часов для более чем 30 компаний. Эти проекты помогли мне набрать хорошую базу знаний и научиться работать с разными командами.

В 2019—2021 годах я присоединился к Andersen Lab. Там руководил командой из 14+ профессионалов и получил награду «Бог UX». Для меня это был важный опыт, когда я осознал, как важно уметь не только создавать классный дизайн, но и вести за собой людей.

С 2021 года участвую в проектах для крупных корпораций. Сейчас у меня уже 8 сертификатов в сфере UX/UI, и я продолжаю учиться, чтобы оставаться полезным и актуальным для своих заказчиков и коллег.

Глава 1: Сдвиг парадигмы: когда AI перестал быть ассистентом и стал со-дизайнером

Есть момент в жизни каждого инструмента, когда он перестаёт быть просто инструментом. Молоток остаётся молотком, сколько бы ты им ни пользовался. Но иногда происходит что-то другое: технология пересекает какую-то невидимую черту и начинает вести себя не как продолжение твоей руки, а как продолжение твоей мысли. Именно это случилось с AI в области дизайна где-то между 2024 и 2026 годами, и если ты не заметил этот момент, то, скорее всего, ты сейчас читаешь эту книгу именно потому, что почувствовал его последствия на своей шкуре.

В первой главе мы говорили о том, что ИИ меняет профессию дизайнера на структурном уровне, а не просто ускоряет рутину. Теперь пришло время разобраться в механике этого изменения. Не в абстракции, а в конкретике: что именно произошло, почему это произошло именно сейчас и, самое важное, как это меняет правила игры для людей с реальным опытом и насмотренностью. Потому что парадокс этого сдвига в том, что он одновременно угрожает опытным дизайнерам и даёт им суперсилу, которую не достать никаким джуном, освоившим Midjourney за выходные.

Эта глава про то, как читать новый ландшафт. Не для того, чтобы испугаться, и не для того, чтобы успокоиться. А для того, чтобы принимать точные решения о том, куда двигаться дальше.

От автодополнения к автономии: что изменилось в 2025—2026 годах

Первое поколение AI-инструментов для дизайна было, если честно, довольно скучным. Оно помогало заполнять placeholder-текст, предлагало варианты цветовых палитр, генерировало иконки по ключевым словам и ускоряло поиск референсов. Полезно, да. Революционно? Нет. Это был умный автокомплит, который экономил тебе полчаса в день и при этом требовал столько же времени на исправление своих же ошибок. Дизайнеры тех лет часто описывали работу с AI как игру в пинг-понг с партнёром, который бьёт по мячу хаотично: иногда прямо в точку, чаще мимо сетки.

То, что произошло в 2025—2026 годах, принципиально отличается по своей природе. Модели получили несколько вещей одновременно: длинный контекст, позволяющий удерживать в памяти целый дизайн-проект со всеми его ограничениями и решениями; мультимодальность, дающую возможность видеть и анализировать визуальные артефакты наравне с текстом; и, что важнее всего, способность к рассуждению, которое в профессиональном сообществе стали называть chain-of-thought reasoning. Проще говоря, AI начал не просто отвечать на вопрос, а думать над ним. Не имитировать мышление, а реально проходить через этапы анализа, которые прежде были доступны только человеку.

Для дизайнера это означает принципиально другой тип диалога. Раньше ты давал команду и получал результат, как в поисковике. Теперь ты ставишь задачу и получаешь контрвопросы, уточнения, предложения рассмотреть альтернативный фрейминг проблемы. Если ты просишь Claude разработать навигацию для финтех-приложения, он не просто генерирует варианты меню. Он спрашивает о частоте использования разных функций, замечает противоречие между двумя твоими требованиями, предлагает рассмотреть прогрессивное раскрытие сложности как паттерн. Это не автодополнение. Это собеседник, у которого есть мнение.

Отдельно стоит поговорить о том, что произошло с агентными возможностями. В 2024 году агентный AI был скорее концептом, чем реальностью: системы могли выполнять цепочки действий, но требовали постоянного надзора и часто ломались на третьем шаге. К 2026 году пайплайны стали достаточно стабильными, чтобы делегировать им реальные задачи без страха прийти утром к сломанному воркфлоу. Дизайнер теперь может задать направление движения, уйти на обед и вернуться к набору вариантов, готовых к оценке.

Почему опытные дизайнеры проигрывают джунам, не освоившим AI — и как это остановить

Давай проговорим неудобную вещь, которую многие предпочитают обходить эвфемизмами. Сегодня джун с хорошим чутьём на промпты и полугодовым опытом работы с AI-инструментами может производить артефакты, которые внешне неотличимы от работы мидла с тремя годами стажа. Может быстрее. Может дешевле. Это не теоретическая угроза из LinkedIn-поста с красным фоном, это реальность, о которой рекрутеры крупных продуктовых компаний говорят вполголоса за закрытыми дверями.

Но вот что важно понять: это не потому, что опытные дизайнеры плохи. Это потому, что опыт создаёт ловушку. Когда ты несколько лет оттачивал конкретный навык рисования в Figma, настройки компонентов или построения прототипов, у тебя формируется сильная мышечная память на определённый процесс. И когда приходит инструмент, который делает этот процесс неактуальным, первая реакция опытного человека, как правило, защитная: он находит в новом инструменте недостатки, которые реальны, но не являются приговором. Джун такой памяти не имеет. Он просто берёт то, что работает, и идёт дальше. Психологи назвали бы это преимуществом отсутствия sunk cost, которого нет у новичка.

При этом опытный дизайнер обладает тем, чего у джуна нет и не может быть быстро: пониманием того, почему что-то работает, а что-то нет. Он знает, что когнитивная нагрузка убивает конверсию не абстрактно, а в конкретном контексте конкретной аудитории. Он чувствует, когда визуальная иерархия лжёт пользователю, даже если формально всё выглядит правильно. Он умеет читать стейкхолдеров и превращать их противоречивые запросы в работающий продукт. Это и есть то самое дизайн-мышление, которое AI воспроизвести не может, потому что оно живёт в пространстве смыслов, а не паттернов.

Проблема в том, что это преимущество теряет ценность, если его не упаковать в AI-ускоренный воркфлоу. Твоя насмотренность стоит дорого только тогда, когда ты можешь материализовать её в артефакт быстрее и точнее, чем джун. А для этого нужно перестать воспринимать AI как угрозу своему мастерству и начать воспринимать его как мультипликатор этого мастерства. Конкретный рецепт выглядит так: сначала идентифицировать те части своего процесса, где ты тратишь время на исполнение, а не на мышление, и делегировать их AI. Затем освободившееся время вложить в те зоны, где твой опыт создаёт необратимое преимущество: в стратегию, в принятие решений, в управление качеством.

Карта нового ландшафта: Claude Design, GPT-4o Vision, Gemini и другие игроки

Прежде чем идти дальше, нужно разобраться с географией. Потому что словосочетание «AI для дизайна» сегодня покрывает настолько разные вещи, что разговор без чёткого понимания того, о чём именно мы говорим, быстро превращается в кашу. Есть минимум три уровня, на которых AI входит в дизайн-процесс: инструменты генерации изображений, языковые модели с мультимодальными возможностями и специализированные продуктовые решения поверх этих моделей.

Claude Design, который занимает центральное место в этой книге, относится ко второму и третьему типу одновременно. Это не просто языковая модель, к которой прицепили возможность загружать картинки. Это система, оптимизированная под производство структурированных, интерактивных артефактов, включая HTML-прототипы, дизайн-документацию и кодовые компоненты. Её сильная сторона состоит в сочетании глубокого понимания контекста, высокого качества рассуждений и способности удерживать сложные ограничения на протяжении длинного диалога. Если ты говоришь Claude Design: «Мы делаем продукт для пожилых пользователей с низкой цифровой грамотностью, минимальный кегль 18, никаких иконок без подписей, акцент на тактильную уверенность», он не только принимает это как инструкцию, но и применяет к каждому последующему решению, не дожидаясь напоминания.

GPT-4o Vision от OpenAI играет в схожем пространстве, но с другими акцентами. Его мультимодальные возможности традиционно считаются одними из лучших в классе: он умеет анализировать скриншоты интерфейсов с высокой точностью, идентифицировать компоненты, описывать визуальную иерархию и предлагать улучшения. Для задач, где нужно взять существующий UI и понять, что с ним не так, GPT-4o Vision остаётся сильным выбором. При этом его слабость состоит в меньшей предсказуемости на длинных сессиях и более ограниченном контексте по сравнению с Claude при работе с комплексными дизайн-системами.

Gemini от Google занимает интересную нишу благодаря нативной интеграции с рабочими инструментами Google и сильным возможностям работы с данными. Для дизайнеров, чей процесс плотно завязан на Google Workspace и чьи проекты требуют активной работы с метриками и пользовательскими данными, Gemini предлагает связность, которую конкуренты могут воспроизвести только через интеграции. Отдельно стоит упомянуть Midjourney и Stable Diffusion как инструменты генерации визуальных референсов, но здесь важно быть честным: это не AI-партнёры по дизайн-мышлению, это очень мощные генераторы изображений. Они дают тебе референс, но не концепцию. Путать эти вещи, означает неправильно распределять время и ожидания.

На горизонте 2026 года активно развивается ещё один класс игроков, которых стоит держать в голове: специализированные AI-нативные дизайн-инструменты, встроенные непосредственно в Figma, Framer и аналогичные среды. Их ценность в том, что они убирают шаг экспорта и импорта между дизайн-средой и языковой моделью. Их ограничение в том, что они, как правило, используют модели в ограниченном режиме, оптимизируя под конкретные сценарии и жертвуя гибкостью. Знать ландшафт целиком, понимать сильные стороны каждого игрока и уметь собирать их в стек под конкретную задачу, это и есть базовая грамотность AI-дизайнера 2026 года.

Как измерить своё AI-преимущество: персональный аудит компетенций

Прежде чем двигаться к конкретным инструментам и техникам, которые разберут последующие главы, имеет смысл сделать паузу и разобраться со своей отправной точкой. Потому что «освоить AI» означает очень разные вещи для человека, который никогда не писал промпт длиннее двух предложений, и для того, кто уже строил агентные пайплайны. Без честного аудита своих компетенций ты рискуешь потратить время на вещи, которые тебе уже не нужны, и пропустить зоны, где у тебя реальные пробелы.

Аудит удобно проводить по четырём измерениям. Первое, уровень работы с промптами: можешь ли ты формулировать задачу так, чтобы получать нужный результат с первой-второй итерации, или каждый раз идёшь методом проб и ошибок через пять-семь ходов? Второе, качество управления контекстом: умеешь ли ты удерживать сессию на протяжении сложной задачи, структурировать входящий контекст так, чтобы модель не теряла нить, и знаешь ли ты, когда нужно начать новую сессию вместо того, чтобы тащить деградирующий контекст дальше? Третье, интеграция в реальный воркфлоу: AI у тебя это отдельная вкладка, в которую ты заглядываешь иногда, или он встроен в твой процесс так, что ты не представляешь, как работал без него? Четвёртое, критическое мышление по отношению к AI-выводу: умеешь ли ты быстро идентифицировать ошибки, галлюцинации и субоптимальные решения, которые модель иногда выдаёт с высокой уверенностью?

Честные ответы на эти вопросы дадут тебе не оценку, а карту. Если ты силён в первом и слаб в четвёртом, это означает конкретную угрозу: ты умеешь быстро генерировать, но принимаешь AI-вывод некритично, что в долгосрочной перспективе подорвёт качество твоей работы сильнее, чем неумение пользоваться AI вообще. Если ты хорошо справляешься с управлением контекстом, но AI существует у тебя в отдельной вкладке, это говорит о том, что ты понимаешь инструмент, но не изменил под него архитектуру своего рабочего процесса, а значит, получаешь от него двадцать процентов потенциальной ценности.

Последнее, что важно сказать об аудите: его результаты нужно обновлять примерно раз в квартал. Не потому что ты так быстро деградируешь, а потому что ландшафт меняется с такой скоростью, что компетенция, которая делала тебя продвинутым пользователем в январе, к апрелю может стать базовым ожиданием рынка. Это не паранойя, это просто честное описание того, в каком темпе сейчас движется индустрия. Умение спокойно принять этот темп и встроить регулярный аудит в свою практику, само по себе является одним из ключевых навыков AI-дизайнера нового поколения.

Контрольные вопросы

Почему именно опытные дизайнеры оказываются наиболее уязвимы перед AI-трансформацией, и какой психологический механизм лежит в основе этой уязвимости?

Какие конкретные зоны дизайн-процесса создают необратимое преимущество для опытного специалиста, которое AI не может воспроизвести, и как правильно защищать и монетизировать это преимущество?

Почему способность критически оценивать AI-вывод может быть важнее, чем умение писать эффективные промпты, и как это влияет на долгосрочное качество работы?

Как правильно описать рекрутерам и стейкхолдерам ценность своего опыта в мире, где джун с AI-инструментами способен воспроизвести значительную часть артефактов старшего дизайнера?

Глава 2: Claude Design: анатомия инструмента, который напугал Adobe на 6%

Когда в главе о сдвиге парадигмы мы разговаривали о том, что AI перестал быть ассистентом и стал со-дизайнером, это звучало как метафора. Красивая, полезная, но всё же метафора. Теперь пора перейти к конкретике: к инструменту, который сделал эту метафору операциональной. Claude Design появился в публичном поле без громких анонсов и маркетинговых ивентов, которые принято устраивать, когда хочешь произвести впечатление. Он просто начал работать. И в течение нескольких недель после его появления акции Adobe упали на шесть процентов. Не потому что инвесторы запаниковали без причины, а потому что кто-то посмотрел на то, что умеет Claude Design, сравнил с тем, за что Adobe берёт деньги, и сделал несложный вывод.

Эта глава про анатомию. Не про вдохновение и не про то, как AI изменит мир. Про то, из чего состоит инструмент, как он устроен изнутри, где у него действительно сильные стороны, а где существуют ограничения, о которых производитель предпочитает не говорить в первых строках документации. Если ты собираешься использовать Claude Design как профессиональный инструмент, а не как игрушку для экспериментов, тебе нужна эта карта.

Что такое Claude Design на самом деле — и чем он не является

Начнём с демонтажа нескольких устойчивых заблуждений, которые вокруг Claude Design успели сформироваться быстрее, чем большинство людей успело им воспользоваться. Первое заблуждение: Claude Design, это Midjourney для интерфейсов. Midjourney генерирует изображения, которые выглядят как интерфейсы, но не являются ими функционально. Claude Design генерирует код, HTML, CSS, JavaScript, который работает в браузере, реагирует на взаимодействие пользователя и может быть непосредственно передан в разработку. Разница примерно такая же, как между фотографией скрипки и самой скрипкой.

Второе заблуждение: Claude Design, это просто Claude, которому показали много макетов Figma. Это тоже неверно, хотя и ближе к реальности. Claude Design, это специализированная конфигурация модели Anthropic с расширенными возможностями для создания интерактивных артефактов, обогащённая дизайн-специфичными инструкциями, систем-промптами и пайплайнами, которые переводят намерение дизайнера в работающий код с уровнем точности, недоступным через стандартный интерфейс Claude. Это имеет значение, потому что меняет то, как ты взаимодействуешь с инструментом: как с системой, которая понимает дизайн-намерение и умеет его реализовать.

Третье и самое важное заблуждение: Claude Design, это замена Figma. Вот здесь нужно быть особенно точным, потому что именно это заблуждение приводит либо к разочарованию, либо к недоиспользованию инструмента. Claude Design работает в другом месте пайплайна. Figma, это среда для проработки, итерации и коммуникации дизайн-решений внутри команды. Claude Design, это среда для быстрой материализации идей в виде работающих прототипов, для исследования вариантов с непропорционально низкими временными затратами и для создания production-близкого кода из дизайн-намерения. Они дополняют друг друга, а не конкурируют. Компании, которые поняли это раньше других, уже выстроили пайплайны, где оба инструмента занимают свои законные места.

Так что же такое Claude Design на самом деле? Это мультимодальная система создания интерактивных дизайн-артефактов, которая принимает на вход текст, изображения, скриншоты и кодовые фрагменты, а на выходе производит работающие интерфейсы с логикой взаимодействия, анимациями и адаптивным поведением. Она понимает дизайн-системы, если ты правильно их ей объяснил. Она знает о принципах доступности и применяет их по умолчанию, если не попросить об обратном. Она умеет работать в нескольких стилевых регистрах и миксировать их по твоему запросу. И что самое важное, она умеет итерировать: принимать обратную связь на естественном языке и трансформировать её в конкретные изменения в коде и визуале, сохраняя при этом контекст предыдущих решений. Именно это сочетание и напугало Adobe: отдельная функция уступает место системе, которая закрывает несколько болевых точек одновременно.

Opus 4.7 vs 4.8: как выбрать модель под конкретную задачу

Разговор о выборе модели часто превращается в технический снобизм: люди перечисляют бенчмарки, сравнивают токены и делают вид, что это единственное, что имеет значение. Для дизайнера это неверный подход. Правильный вопрос не «какая модель лучше», а «какая модель лучше для конкретного типа задачи». И здесь разница между Opus 4.7 и Opus 4.8 оказывается принципиальной, хотя она далеко не очевидна из официальной документации.

Opus 4.7, это модель, которую имеет смысл использовать для задач, где важна последовательная, глубокая проработка одного решения. Если ты строишь сложный компонент с нетривиальной логикой состояний, проектируешь дизайн-систему с нуля или работаешь с большим контекстом, где нужно удерживать множество взаимосвязанных ограничений одновременно, 4.7 даст более стабильный и предсказуемый результат. Она медленнее, это правда. Но в задачах, где качество рассуждения имеет критическое значение, эта скорость является артефактом более тщательного процесса.

Opus 4.8 работает иначе. Её архитектурные изменения направлены прежде всего на ускорение итерационного цикла и улучшение работы с мультимодальными входными данными. Если ты исследуешь варианты, быстро перебираешь стилевые направления, работаешь со скриншотами и визуальными референсами или ведёшь диалог с несколькими итерациями в быстром темпе, 4.8 будет значительно эффективнее. Её отклик на визуальный ввод точнее, она лучше удерживает стилевую последовательность в рамках одной сессии и демонстрирует заметно улучшенное понимание пространственных отношений в интерфейсе, что критически важно для работы с макетами.

Практическое правило, которое сложилось у большинства профессиональных пользователей, работающих с обеими версиями, выглядит примерно так. Ранняя стадия проекта, когда нужно быстро проверить несколько концептуальных направлений, работа с визуальными референсами и стилевыми экспериментами, быстрые итерации в диалоговом режиме, это территория 4.8. Финальная проработка компонента, который пойдёт в систему, глубокий аудит дизайн-решения на соответствие ограничениям, работа с большими кодовыми контекстами, это территория 4.7. Переключаться между ними внутри одного рабочего дня не только допустимо, но и разумно. Воспринимай их как разные кисти, каждая для своей поверхности: вопрос в том, для какой поверхности ты работаешь сейчас.

Artifacts, Canvas и мультимодальный ввод: полная карта возможностей

Когда смотришь на интерфейс Claude Design в первый раз, он кажется обманчиво простым. Поле для ввода, область вывода, несколько кнопок. Это обманчивая простота, за которой скрывается значительно более богатая система, и большинство пользователей обнаруживают её возможности случайно, а не системно. Давай пройдём по этой карте намеренно.

Artifacts, это базовый строительный блок работы с Claude Design. Артефакт, это любой самостоятельный выходной объект, который система создаёт и который может быть использован независимо от чата: HTML-страница, компонент на React, SVG-иллюстрация, JSON-конфиг дизайн-системы, интерактивный прототип с логикой переходов. Артефакты существуют как отдельные объекты, которые можно редактировать напрямую, передавать дальше по пайплайну или экспортировать. Ключевое свойство артефакта: он живёт в отдельном пространстве от диалога, что позволяет итерировать, не теряя историю разговора и контекст предыдущих решений. Это звучит как техническая деталь, но на практике меняет весь характер работы: ты больше не получаешь одноразовые ответы, ты строишь объекты.

Canvas, это режим работы, который стоит воспринимать как пространство совместного редактирования. Если Artifacts, это создание объекта, то Canvas, это работа с ним вместе с системой. В режиме Canvas ты можешь указывать на конкретные элементы интерфейса, просить изменить только их, не трогая остальное, комментировать отдельные части макета и получать точечные правки вместо полной перегенерации. Для дизайнера, привыкшего к работе в Figma, это наиболее близкая по ощущению механика: ты работаешь с конкретным объектом прямо в интерфейсе. Canvas особенно полезен на стадии финальных итераций, когда общая структура уже утверждена и нужно работать с деталями.

Мультимодальный ввод, это то место, где Claude Design получает наиболее значительное конкурентное преимущество перед большинством альтернатив. Система принимает несколько типов входных данных одновременно: текст, скриншоты и фотографии экрана, загруженные файлы изображений, URL для анализа веб-страниц, фрагменты кода. Принимает и умеет работать с их комбинациями. Ты можешь дать скриншот чужого интерфейса с текстовым описанием того, что тебе в нём нравится и что нет, добавить URL своего существующего продукта как референс фирменного стиля и попросить создать новый экран, который будет решать конкретную задачу в стилевой логике первого и с улучшениями относительно второго. Речь идёт об одном промпте, трёх источниках контекста и одном решении на выходе. Именно это делает мультимодальный ввод принципиально другим способом работы с инструментом. Важно понимать, что когда Claude Design интерпретирует визуальные референсы и воспроизводит типографические решения, он оперирует теми же базовыми принципами, которые Эллен Луптон систематизировала в «Thinking with Type»: иерархия, ритм, пространство и отношения между буквенными формами как инструменты коммуникации, а не просто декорации. Луптон показала, что типографика представляет собой саму структуру смысла, и именно это понимание система применяет, когда переводит визуальный ввод в работающий интерфейсный код.

Лимиты, тарифы и обходные пути: Pro, Max, Team, Enterprise

Поговорим о деньгах. Неправильный выбор тарифа в случае с Claude Design стоит либо реальных денег, либо реальной производительности, и часто одновременно того и другого. Антропик выстроил свою тарифную линейку логично, но с несколькими неочевидными нюансами, которые в официальной документации спрятаны мелким шрифтом.

Pro за двадцать долларов в месяц — это точка входа, но она не так ограничена, как кажется на первый взгляд. Ты получаешь доступ к обеим моделям, Opus 4.7 и Opus 4.8, к полному функционалу Artifacts и к мультимодальному вводу. Лимит на использование обновляется каждые восемь часов, и если ты работаешь как большинство дизайнеров, то есть активно утром, потом встречи, потом снова вечером, то в реальности ты упираешься в потолок крайне редко. Проблема начинается, когда ты используешь Claude Design для тяжёлых итерационных задач: взял крупный компонент, делаешь двадцать правок подряд, получаешь рефакторинг каждого варианта. Вот здесь Pro начинает дышать в затылок, и ты обнаруживаешь, что конец дня встречаешь с заблокированным интерфейсом именно тогда, когда нужно было сдать прототип.

Max за сорок долларов убирает это напряжение практически полностью. Главное отличие здесь заключается в объёме: лимиты на использование выше в пять раз по сравнению с Pro. Дизайнер, использующий Claude Design как основной инструмент прототипирования, а не как периодический помощник, почувствует эту разницу принципиально. Представь, что ты ведёшь дизайн нового онбординга для SaaS-продукта с нуля: тебе нужно собрать пять экранов, каждый в двух-трёх вариантах, потом быстро показать стейкхолдерам, потом переделать после правок. Это легко сорок-пятьдесят качественных запросов за рабочий день, и Max здесь просто снимает вопрос с повестки. Дополнительно Max даёт приоритет в очереди в периоды пиковой нагрузки на серверы, что на практике означает разницу в скорости ответа от трёх до пятнадцати секунд в часы, когда Anthropic загружен, а именно в середине рабочего дня по американскому времени.

Team стоит двадцать пять долларов на пользователя в месяц при минимуме пяти мест, и здесь логика уже совсем другая. Ключевая ценность заключается в административных возможностях: единое рабочее пространство, общие промпты и контексты, которые не нужно воссоздавать каждому участнику заново, и отсутствие обучения модели на данных вашей организации, что критично, если ты работаешь с проектами под NDA или клиентским IP. Для дизайн-команды из шести человек это означает, что тимлид один раз пишет системный промпт с контекстом продукта, дизайн-принципами и ограничениями, закрепляет его в рабочем пространстве, и каждый участник начинает разговор с моделью уже с пониманием, что это за продукт и как в нём принято думать. Экономия на онбординге новых членов команды и на поддержании консистентности выходит ощутимая.

Enterprise разговор отдельный и честный: это другой уровень контроля над данными, SLA-гарантии, возможность настройки SAML SSO и кастомные условия обработки данных. Студиям, работающим с крупными брендами или регулируемыми индустриями, вроде финансов или медицины, Enterprise закрывает базовые требования безопасности, а не остаётся в категории luxury-опций. Цена здесь договорная, но ориентировочно начинается от ста пятидесяти долларов на пользователя в месяц для небольших команд. Практический совет, который многие упускают: если ты работаешь в агентстве или студии и ведёшь переговоры с Enterprise, проси включить в контракт расширенный лимит на контекстное окно и приоритетный доступ к новым моделям при релизе. Anthropic это предоставляет, но не предлагает сам, и большинство команд обнаруживают эту возможность только тогда, когда контракт уже подписан.

Контрольные вопросы

Чем принципиально отличается Claude Design от инструментов генерации изображений вроде Midjourney, и почему это различие важно для профессионального дизайнера?

Что такое Artifacts в контексте Claude Design и почему их природа как отдельных объектов меняет характер работы с инструментом?

Как мультимодальный ввод позволяет комбинировать несколько источников контекста в одном промпте, и какие задачи это делает принципиально более эффективными?

Чем режим Canvas отличается от стандартного режима работы с артефактами, и в какой момент дизайн-процесса он наиболее ценен?

Глава 3: Промпт как дизайн-документ: почему ваш текст важнее вашего вкуса

Анатомия инструмента, которую мы разобрали в предыдущей главе, даёт понимание того, что именно перед тобой стоит. Но понимание устройства механизма и умение им управлять, это разные вещи. Можно знать, как работает двигатель внутреннего сгорания, и при этом не уметь водить машину. С Claude Design та же история: знание того, что такое Artifacts и чем Opus 4.8 отличается от 4.7, не сделает тебя автоматически человеком, который получает нужный результат с первого раза. Это делает промпт.

Промпт, это не поисковый запрос и не команда. Это проектный документ. Он содержит не просто задание, а контекст, ограничения, намерения, критерии качества и, если ты делаешь всё правильно, встроенные механизмы итерации. Большинство дизайнеров, которые жалуются на то, что Claude «не понимает, чего я хочу», в действительности сталкиваются не с ограничением модели, а с собственным неумением формулировать задачу так, как её понял бы умный, но лишённый телепатии коллега. И это, при всей кажущейся банальности, меняет всё.

Эта глава про то, как писать промпты, которые работают. Не потому что в них есть магические слова или секретные ключевые фразы, а потому что они устроены так же, как хороший дизайн-документ: они дают достаточно контекста, чтобы принимать правильные решения на каждом развилке, и достаточно пространства, чтобы эти решения не были механическим следованием инструкции. Разница между дизайнером, который борется с AI, и дизайнером, который работает с AI, чаще всего умещается в несколько абзацев текста.

Структура промпта, который даёт результат с первого раза

Начнём с разрушения одного устойчивого мифа: хороший промпт, это длинный промпт — заблуждение. Длина сама по себе ничего не гарантирует. Можно написать пятьсот слов контекста и получить на выходе ровно то, что Claude выдал бы на двадцати словах без него. А можно написать три чётких предложения и получить артефакт, который нужно лишь слегка подкрутить. Результат определяет структура. Точнее, то, присутствуют ли в тексте все элементы, которые модели нужны для принятия решений.

Этих элементов пять, и их можно думать как слои. Первый слой, это роль и контекст. Claude должен понимать, кто ты в этой задаче и для чего она решается. Речь идёт о реальном операционном контексте: «я проектирую онбординг для B2B SaaS-продукта, аудитория, финансовые менеджеры среднего звена в компаниях от 50 до 500 человек, они технически грамотны, но не разработчики». Это даёт модели карту местности, по которой предстоит двигаться. Второй слой, задача, сформулированная в терминах результата, а не процесса. Формулировка «создай форму регистрации, после прохождения которой пользователь должен почувствовать, что сложность продукта управляема и стоит его времени» работает иначе, чем просто «создай форму регистрации». Разница между этими формулировками, это разница между техническим заданием и дизайн-намерением.

Третий слой, ограничения. Речь идёт о реальных границах, внутри которых должно существовать решение: технический стек, доступность, существующая дизайн-система, корпоративные гайдлайны, конкретные компоненты, которые уже используются и не подлежат переработке. Четвёртый слой, критерии оценки. Этот слой пропускают почти все, и именно его отсутствие приводит к тому, что Claude делает «что-то хорошее», но не обязательно то, что нужно. Если ты формулируешь, по каким признакам результат будет считаться успешным, «форма должна умещаться в один экран без скролла на 1366px, содержать не более семи полей и заканчиваться микрокопией, которая снижает тревогу перед нажатием кнопки»,, ты даёшь модели внутренний фильтр качества. Пятый слой, формат вывода. Укажи, что именно ты хочешь получить: HTML с инлайновыми стилями, React-компонент с Tailwind, структурированный список с пояснениями к каждому решению, или интерактивный артефакт. Без этого указания Claude делает выбор за тебя, и он не всегда совпадает с твоими ожиданиями.

Эти пять слоёв не требуют строгого порядка и не обязаны присутствовать в виде размеченных секций. Лучшие промпты, это связный текст, в котором все пять элементов присутствуют органично, как если бы ты объяснял задачу умному коллеге на брифинге перед работой. Попробуй проверить свой следующий промпт по простому критерию: если бы ты отправил его человеку, который никогда не видел твой продукт и не знает твоего вкуса, смог бы он принять все ключевые дизайн-решения самостоятельно и получить что-то близкое к тому, что ты имел в виду? Если нет, какого именно слоя не хватает?

Референс вместо тысячи слов: техника скриншот-инъекции

Есть категория дизайн-намерений, которую словами передать почти невозможно. Визуальный опыт работает иначе, чем вербальный. Когда ты говоришь «минималистичный», ты имеешь в виду что-то конкретное, сложившееся из десятков насмотренных примеров. Когда Claude читает слово «минималистичный», он интерпретирует его через статистику своих обучающих данных, которая может существенно расходиться с твоим внутренним определением. Именно здесь скриншот-инъекция становится принципиальным методом работы.

Техника проста по механике и сложна по применению. Ты загружаешь в диалог изображение, скриншот существующего интерфейса, страницу из портфолио, фотографию физического объекта, который даёт нужное настроение, или даже рукописный скетч, и используешь его как опорную точку в формулировке задачи. Ключевой момент здесь, точность указания на то, что именно из референса ты хочешь перенести, а что намеренно оставляешь за бортом. Плохая скриншот-инъекция выглядит так: «сделай как здесь» с прикреплённым изображением. Хорошая выглядит так: «посмотри на эту карточку продукта, меня интересует принцип работы с типографской иерархией и использование негативного пространства вокруг CTA, при этом цветовая схема и иконографика не релевантны». Ты извлекаешь из референса конкретные дизайн-решения и применяешь их в новом контексте, прося Claude работать с выбранными элементами, а не воспроизводить референс целиком.

Особенно мощной эта техника становится при работе с несколькими референсами одновременно. Два-три изображения, каждое из которых иллюстрирует отдельный аспект желаемого результата, дают модели гораздо более точное определение пространства решений, чем любой текстовый дескриптор. Представь, что ты объясняешь задачу дизайнеру через mood board: один скриншот показывает нужную плотность информации на экране, другой, тональность микрокопии, третий, принцип анимации при переходах. Вместе они образуют интерсекцию, которая описывает желаемый результат точнее, чем каждый из них по отдельности. Добиться той же точности словами было бы значительно труднее и дольше.

Есть ещё одно применение скриншот-инъекции, о котором редко говорят: загрузка собственного существующего дизайна как точки отсчёта. Если ты итерируешь поверх уже созданного экрана, показываешь Claude то, что есть, и формулируешь, что именно нужно изменить и почему, ты экономишь огромное количество контекстной работы. Модель видит реальное состояние дизайна, а не твоё описание этого состояния, и может работать с конкретными проблемами, а не с гипотетическими. Скриншот-инъекция становится инструментом для умной итерации существующего, а не только техникой для создания нового.

Итерационный диалог: как вести Claude к нужному решению за 2—3 хода

Одно из самых распространённых заблуждений об эффективной работе с AI, это представление о том, что профессионализм измеряется способностью получить идеальный результат за один промпт. Это ложная метрика, которая заставляет дизайнеров тратить по двадцать минут на формулировку первого сообщения вместо того, чтобы использовать то, что делает Claude по-настоящему мощным инструментом: его способность к контекстуальному диалогу. Правильная работа с Claude, это управляемое сближение с целью за несколько итераций.

Для того чтобы диалог был управляемым, а не хаотичным, важно понимать принцип прогрессивного уточнения. Первый промпт задаёт направление и основные параметры, он намеренно не должен быть исчерпывающим. Его задача состоит в том, чтобы получить первый артефакт, который станет видимой точкой для разговора. Многие дизайн-решения невозможно сформулировать абстрактно, их нужно увидеть, чтобы понять, чего именно не хватает. Поэтому первая итерация, это попытка материализовать направление. Второй промпт работает с конкретным артефактом. Он указывает на то, что работает и должно быть сохранено, и на то, что не работает и нуждается в изменении, с обязательным объяснением причины. Формулировка «заголовок слишком большой» менее полезна, чем «заголовок слишком большой, потому что в контексте таблицы с данными он создаёт ощущение, что страница принадлежит маркетингу, а не аналитическому инструменту». Причина, это всегда часть инструкции.

Третий промпт, если он нужен, обычно занимается точной калибровкой: деталями, которые стали видны только после второй итерации. Это может быть состояние hover на кнопке, которое не совпадает с тональностью остального интерфейса, или отступ, который нарушает ритм сетки. К третьей итерации у тебя должно быть ощущение, что ты полируешь, а не переделываешь. Если этого ощущения нет и ты снова меняешь направление, это сигнал того, что ты сам не определился с тем, чего хочешь. В этом случае правильнее сделать шаг назад и переосмыслить задачу, а не продолжать итерировать в надежде, что нужное решение появится случайно.

Есть практический приём, который делает итерационный диалог значительно эффективнее: в конце каждого промпта, начиная со второго, фиксируй то, что уже решено и что менять не нужно. Это звучит как избыточная осторожность, но на практике это защита от очень распространённой ошибки, которую совершают даже опытные пользователи: Claude в процессе внесения изменений может «переоптимизировать» часть, которая тебя устраивала, потому что счёл её несогласованной с новыми изменениями. Явная фиксация того, что не трогать, создаёт контракт между тобой и моделью, который экономит не только время, но и нервы.

Антипаттерны промптинга, которые убивают качество вывода

Разговор о том, как делать правильно, будет неполным без честного разбора того, как делают неправильно. Антипаттерны промптинга, это ошибки, которые совершают в том числе опытные дизайнеры, потому что они кажутся интуитивно правильными. Именно поэтому они устойчивы и именно поэтому их стоит назвать прямо.

Первый и самый распространённый антипаттерн, это эстетические дескрипторы без функционального обоснования. Промпты вроде «сделай красиво», «добавь ощущение премиальности», «должно быть современно и чисто», это инструкции в режиме вкусовщины. Они не дают Claude ничего операционного, потому что «красиво» и «премиально» в разных контекстах означают диаметрально противоположные решения. Fintech-премиальность выглядит иначе, чем luxury fashion, которая выглядит иначе, чем enterprise SaaS. Каждый раз, когда ты ловишь себя на использовании чисто эстетического дескриптора, спрашивай себя: а что именно я имею в виду под этим словом в контексте этой конкретной задачи? Ответ на этот вопрос и есть твой промпт.

Второй антипаттерн, это противоречивые требования, которые дизайнер не замечает сам. «Сделай минималистично, но чтобы все функции были на виду и пользователь сразу понимал все возможности», это технически невыполнимая задача, и Claude будет вынужден принять за тебя решение о том, какое из требований важнее. Иногда это решение оказывается правильным, но чаще, нет. Если ты обнаруживаешь в своём промпте «но», «при этом», «однако», это сигнал остановиться и проверить, не конфликтуют ли части требования между собой. Дизайн-компромисс должен быть осознанным, а не случайным.

Третий антипаттерн, это избыточная вежливость и хеджирование, которые размывают задачу. Промпты вида «может быть, попробуй как-нибудь сделать что-то вроде…» или «не знаю, насколько это возможно, но было бы здорово, если бы…» сигнализируют модели о неуверенности в задаче. Claude устроен таким образом, что неопределённость в запросе он склонен компенсировать усреднённым, безопасным результатом. Если ты знаешь, чего хочешь, говори об этом прямо. Если не знаешь, признай это явно и попроси помочь с определением направления, а не маскируй неопределённость вежливыми оговорками. Четвёртый антипаттерн, это отсутствие негативных ограничений. Большинство промптов говорят о том, что должно быть. Лучшие промпты также говорят о том, чего не должно быть. «Без модальных окон», «без carousel-компонентов», «без анимации, которая требует пользовательского действия для остановки», это конкретные, верифицируемые ограничения, которые исключают целые классы нежелательных решений и позволяют Claude сосредоточиться на том пространстве, которое тебе действительно нужно. Пятый антипаттерн, самый тонкий, и именно поэтому самый опасный. Это принятие первого правдоподобного результата без критической проверки. Когда Claude выдаёт убедительно выглядящий артефакт, возникает соблазн счесть задачу выполненной. Но «выглядит правдоподобно» и «решает задачу», слова с разным смыслом. Профессиональная работа с AI требует того же критического взгляда на результат, который ты применяешь к работе любого другого дизайнера: что здесь сделано верно, что неверно, и почему. Без этой проверки ты не управляешь процессом, ты наблюдаешь за ним.

Контрольные вопросы

Чем принципиально отличается формулировка задачи через результат от формулировки через процесс — и как эта разница влияет на дизайн-решения, которые принимает Claude?

В каких ситуациях скриншот-инъекция даёт принципиальное преимущество перед текстовым описанием — и что нужно явно указывать при работе с визуальными референсами, чтобы модель не скопировала нежелательные элементы?

Что такое «негативные ограничения» в промпте и почему их отсутствие заставляет Claude исследовать нежелательное пространство дизайн-решений?

Как фиксация «что не трогать» в промпте второй и третьей итерации предотвращает конкретную и распространённую ошибку при внесении изменений в существующий артефакт?

Глава 4: Brand State Architecture: как заставить AI помнить вашу дизайн-систему

Умение писать промпты, которые дают результат с первого раза, и умение вести итерационный диалог за два-три хода, это мощные инструменты. Но у них есть один общий изъян, о котором в большинстве руководств предпочитают не говорить вслух. Каждый раз, когда ты открываешь новую сессию с Claude, ты начинаешь с чистого листа. Не ты, он. Весь контекст, который ты выстраивал в предыдущем разговоре: цветовые решения, типографика, паттерны взаимодействия, тональность, логика компонентов, всё это испаряется. Claude не помнит, что твой бренд использует 8-пиксельную сетку и никогда не применяет тени с размытием больше 12 пикселей. Он не помнит, что основной цвет, это не просто синий, а конкретный синий с историей и смыслом. Он не помнит вообще ничего.

Эта глава о том, как решить именно эту проблему. Не обходными маневрами в виде повторяющегося вступительного текста в начале каждого промпта, а системно, через архитектуру, которую я называю Brand State Architecture. Это не термин из академического словаря и не маркетинговое название очередного SaaS-продукта. Это описание конкретного подхода: как организовать знание о своей дизайн-системе таким образом, чтобы AI мог к нему обращаться в любой момент, в любой сессии, с любым уровнем детализации. Четыре раздела этой главы проведут тебя от понимания проблемы через технические решения к конкретным артефактам, которые можно создать сегодня вечером и использовать завтра утром.

Проблема «амнезии»: почему AI забывает бренд между сессиями

Чтобы по-настоящему понять проблему, нужно разобраться в её архитектурной природе. Большие языковые модели, включая Claude, работают в рамках так называемого контекстного окна. Это ограниченный объём текста, который модель «видит» в один момент времени и на основе которого формирует ответ. Всё, что находится внутри этого окна, существует для модели. Всё, что за его пределами, не существует вообще. Когда сессия заканчивается, контекстное окно очищается. Следующая сессия начинается с нуля, без каких-либо следов предыдущей. Это фундаментальное свойство архитектуры, которое имеет свои веские технические причины. Проблема в том, что дизайн-системы по своей природе существуют во времени: они накапливаются, уточняются, эволюционируют. И дизайнер, который работает с AI без решения проблемы амнезии, каждый раз тратит значительную часть своего контекстного окна на то, чтобы заново объяснять AI, с кем он разговаривает.

Оцени масштаб потерь конкретно. Допустим, у тебя есть дизайн-система средней зрелости: основная и акцентная палитры с вариациями, типографическая шкала из восьми ступеней, базовый набор из сорока компонентов с вариантами, несколько задокументированных паттернов взаимодействия и принципы работы с отступами. Чтобы передать это Claude в одном промпте достаточно полно для осмысленной работы, тебе понадобится несколько тысяч токенов только на описание системы, ещё до того, как ты сформулируешь реальную задачу. Это структурная проблема, которая делает глубокую работу с AI в рамках установленного бренда непрактичной при традиционном подходе. Ты тратишь лимит контекстного окна на ввод, а не на вывод.

Есть ещё более тонкий аспект этой проблемы, который редко обсуждают. Когда дизайнер каждый раз заново описывает свою дизайн-систему в произвольном тексте, он неизбежно описывает её по-разному. Сегодня он упоминает, что компания использует «спокойный, уверенный визуальный язык». Завтра скажет «минималистичный профессиональный стиль». Послезавтра напишет «корпоративный, но не скучный». Каждое описание задаёт чуть другие координаты в пространстве возможных интерпретаций, и AI каждый раз выдаёт чуть другой результат. Разные входные данные порождают разные результаты. Непоследовательный ввод даёт непоследовательный вывод. Проблема амнезии представляет собой проблему памяти и проблему воспроизводимости.

Понимание этого сдвигает постановку задачи. Задача состоит в том, чтобы создать систему, в которой правильный контекст доставляется к каждому запросу автоматически, последовательно и в машиночитаемом формате. Именно это и есть Brand State Architecture: обход проблемы памяти через правильно спроектированную инфраструктуру контекста. Следующие три раздела, это три уровня этой инфраструктуры, от самого технологичного к самому практичному.

RAG-пайплайн для дизайнера: Pinecone, Weaviate и векторные базы без страха

Аббревиатура RAG расшифровывается как Retrieval-Augmented Generation, и если ты слышишь её впервые или она кажется тебе зоной компетенции исключительно ML-инженеров, то следующие несколько абзацев изменят эту установку. RAG, это технический паттерн, решающий ровно ту проблему, которую мы обсуждали: как давать языковой модели доступ к информации, которая не умещается в её контекстное окно целиком. Принцип работает следующим образом. Твои документы, токены, описания компонентов, гайдлайны, принципы бренда, разбиваются на фрагменты и преобразуются в числовые векторы, которые фиксируют смысловое содержание каждого фрагмента. Эти векторы хранятся в специализированной базе данных. Когда ты задаёшь вопрос или формулируешь задачу, система сначала ищет в векторной базе наиболее релевантные фрагменты и добавляет их в контекст запроса к модели. В результате модель видит не весь твой бренд-гайдлайн разом, а именно ту его часть, которая нужна для конкретного ответа.

Pinecone и Weaviate, два наиболее зрелых инструмента для хранения векторных данных, которые дизайнер может освоить без написания кода с нуля. Pinecone, это облачный сервис с управляемой инфраструктурой: ты создаёшь индекс, загружаешь документы через API или через один из многочисленных готовых коннекторов, и система сама занимается векторизацией и поиском. Pinecone хорошо подходит для ситуаций, когда тебе нужна надёжная работающая система с минимальными настройками. Weaviate, open-source альтернатива с более богатыми возможностями кастомизации и схемой данных, которая позволяет хранить вместе с векторами произвольные метаданные: к примеру, для каждого компонента дизайн-системы ты можешь хранить не только его описание, но и ссылку на Figma-файл, дату последнего обновления, теги контекста использования и имена ответственных. Weaviate требует чуть больше первоначальной настройки, но даёт значительно более гибкую структуру хранения.

Для дизайнера практический вход в мир RAG выглядит как использование готовых инструментов-оберток. Langchain и LlamaIndex, это фреймворки, которые абстрагируют техническую сложность RAG до уровня, с которым справится человек, умеющий читать документацию. LlamaIndex в особенности хорошо подходит для задачи «сделай мою дизайн-документацию доступной для AI»: у него есть встроенные загрузчики для Notion, Confluence, локальных файлов в формате Markdown, PDF и даже веб-страниц. Ты указываешь источник своей дизайн-документации, настраиваешь параметры разбивки на фрагменты и размещения в индексе, и получаешь рабочий пайплайн примерно за час работы, половину из которой займёт чтение документации. Важно: достаточно быть дизайнером, достаточно любопытным, чтобы один раз потратить вечер на настройку системы, которая сэкономит тебе часы каждую неделю.

Есть и ещё более лёгкий вход, который стоит упомянуть отдельно: инструменты типа Cursor AI, Windsurf или Claude Projects позволяют прикреплять файлы и документы к проекту как постоянный контекст. Claude Projects, в частности, хранит загруженные документы и делает их доступными в каждой сессии в рамках проекта, без необходимости настраивать векторную базу самостоятельно. Перед тобой упрощённое решение с семантическим поиском попроще, чем в полноценном RAG, но для большинства практических задач дизайнера это разумный первый шаг. Ты создаёшь проект «Дизайн-система Клиента X», загружаешь туда свои токены, компонентную документацию и гайдлайн, и с этого момента каждая сессия в этом проекте знает, с каким брендом она работает. RAG с Pinecone или Weaviate становится следующим уровнем, когда объём документации вырастает настолько, что даже контекстного окна проекта начинает не хватать или когда тебе нужен тонкий контроль над тем, какие именно фрагменты подставляются в контекст.

Строим папку Claude-Design: токены, палитры, компоненты в machine-readable формате

Если RAG-пайплайн можно сравнить с нервной системой, которая доставляет нужные знания в нужный момент, то то, что мы разберём сейчас, это сам язык, на котором эти знания записаны. Проблема большинства дизайн-систем в том, что они написаны для людей: длинные Notion-страницы с вдохновляющими принципами, Figma-файлы с аккуратно разложенными компонентами, PDF-гайдлайны с красивыми скриншотами. Всё это прекрасно читается на ретина-дисплее за чашкой кофе, но для языковой модели это примерно как пытаться понять чертёж здания, глядя на фотографию его фасада. Форма есть, а структуры нет. Чтобы AI мог работать с твоим брендом как полноценный член команды, а не как гость, который впервые зашёл на сайт, документацию нужно переписать в machine-readable формат. Задача состоит в том, чтобы добавить слой структуры, который модель способна надёжно интерпретировать.

Начнём с самого верхнего и самого недооценённого слоя: бренд-токены. Большинство дизайнеров знакомы с ними в контексте дизайн-систем вроде Figma Tokens или Style Dictionary, но мало кто думает о них как об инструменте общения с AI. А между тем, правильно описанный токен это маленький контракт между брендом и генеративной моделью. Вместо «основной цвет бренда это синий» пиши «primary-500: #1D4ED8, используется для основных интерактивных элементов, включая CTA-кнопки и активные состояния навигации; контрастность с белым фоном составляет 7.2:1, что соответствует WCAG AAA». Разница огромная. Поэзия остаётся в первом варианте, спецификация во втором. Модель, получая второй вариант, уже знает и цвет, и контекст применения, и ограничения доступности одновременно. Когда ты описываешь таким образом пятнадцать-двадцать ключевых токенов из своей системы, у модели формируется достаточно плотная карта бренда, чтобы она могла делать осознанные предложения, а не просто угадывать.

Типографика требует отдельного внимания, потому что это тот слой, где дизайнеры чаще всего грешат неполнотой описания. Написать «используем Inter для интерфейсных текстов и Cormorant Garamond для заголовков лонгридов» недостаточно. Для machine-readable документации тебе нужна таблица ролей: каждому уровню иерархии соответствует конкретный size в пикселях и в rem, конкретный line-height, конкретный letter-spacing и контекст применения. Display-heading: 64px / 4rem, line-height 1.1, letter-spacing minus 0.02em, только для hero-секций и промо-материалов. Body-default: 16px / 1rem, line-height 1.6, letter-spacing 0, для всего основного контента интерфейса. Когда модель видит такую структуру в контексте, она не будет предлагать тебе в UI-копирайтинге задачу использовать Display-heading для подзаголовков карточки. Она понимает иерархию так же, как понимаешь её ты, и это принципиально меняет качество диалога.

Третий слой, паттерны поведения и голос бренда, наименее формализован в большинстве команд, но именно он даёт AI контекст для принятия решений в серых зонах. Поведенческие паттерны это принципы взаимодействия: как бренд реагирует на ошибки пользователя, насколько он разговорчив в пустых состояниях, использует ли юмор в онбординге или держит дистанцию. Записывай это максимально конкретно и, желательно, с примерами антипаттернов. Конкретика решает всё: вместо размытого «мы дружелюбны, но профессиональны» пиши «текст ошибки валидации никогда не начинается со слова «Ошибка:" и никогда не обвиняет пользователя; формат: что пошло не так плюс что нужно сделать, не более двух предложений; пример корректного: «Кажется, email написан неверно. Проверь, есть ли в нём знак @"; пример некорректного: «Неверный формат email»». Это операционная инструкция, и языковая модель работает с ней именно так. Сложив все три слоя, токены, типографику и голос, в единый структурированный документ, который попадает в контекст через RAG или системный

промт, ты получаешь ситуацию, когда AI не просто знает твой бренд, а способен его защищать так же рефлекторно, как это делает опытный дизайнер после двух лет работы в одной команде.

DESIGN.md как центральный файл документации и единый источник правды

Если ты когда-нибудь открывал Notion-страницу своей дизайн-системы через полгода после её создания, то знаешь это ощущение: пятнадцать вложенных разделов, три устаревших варианта цветовой палитры, комментарий от менеджера двухлетней давности с пометкой «разобраться», и где-то в глубине ссылка на Figma-файл, который уже переехал в другой проект. Ты не виноват. Это системная проблема большинства команд: документация живёт в слишком многих местах одновременно, и в итоге не живёт нигде. DESIGN.md решает это радикальным способом, которым программисты пользовались десятилетиями: один текстовый файл, который является единственным авторитетным источником правды о твоём бренде и дизайн-системе. Плоский, читаемый, версионируемый файл, который одинаково хорошо понимают и твой джун-дизайнер в первый рабочий день, и языковая модель в первую секунду нового контекста.

Структура DESIGN.md строится по принципу убывающей абстракции: от самого фундаментального к самому конкретному. Первый раздел описывает суть бренда в терминах, которые нельзя неверно интерпретировать. «Мы дружелюбные и инновационные» здесь не работает, нужно что-то вроде: «Интерфейс обращается к пользователю на „ты“, использует активный залог и избегает пассивных конструкций. Тон варьируется от нейтрально-делового в состояниях ошибки до тепло-поддерживающего в состояниях успеха. Восклицательные знаки используются не чаще одного раза на экран и только в момент значимого достижения пользователя.» Это спецификация поведения, а не поэзия о бренде. Следующий уровень касается визуальных токенов: не просто перечисление цветов, а их семантика. «Primary-600 (#1D4ED8) используется исключительно для интерактивных элементов, требующих основного действия пользователя. Применение этого цвета для декоративных элементов или фоновых поверхностей нарушает семантику системы.» Когда ты пишешь документацию таким образом, ты фактически описываешь то, как система думает. Именно это и нужно языковой модели.

Практическая архитектура файла, которую я рекомендую после нескольких итераций с реальными проектами, включает шесть блоков. Первый: Brand Core, где в пяти-семи предложениях описана суть продукта, его аудитория и позиционирование без маркетинговой воды. Второй: Visual Tokens, полный список дизайн-токенов с именами, значениями и семантическим описанием каждого. Третий: Typography Rules, где указаны правила применения шрифтов и размеров, включая исключения и запрещённые комбинации. Четвёртый: Component Behavior, описание ключевых компонентов через их состояния и логику переходов между ними. Пятый: Accessibility Constraints, обязательные требования к контрасту, размерам тач-зон и альтернативным текстам, сформулированные как жёсткие правила, а не рекомендации. Шестой: Anti-Patterns, раздел, который большинство команд пропускают и о котором потом жалеют: явное описание того, чего делать нельзя и почему. «Не использовать красный цвет для акцентов, не связанных с ошибками или деструктивными действиями» работает лучше, чем любая позитивная инструкция, потому что исключает целый класс ошибок ещё до их появления. Весь файл в зрелом состоянии занимает от восьмисот до полутора тысяч строк и помещается в контекстное окно современных моделей целиком, что делает его идеальным инструментом для прямой работы без RAG-пайплайна на этапах прототипирования.

Главное преимущество DESIGN.md перед любой другой формой документации, которое редко обсуждают, это его версионируемость через Git. Когда ты хранишь файл в репозитории рядом с кодом или в отдельном дизайн-репо, каждое изменение системы становится коммитом с датой, автором и описанием причины изменения. Ты буквально видишь историю эволюции бренда: «02.2024, изменён основной радиус скругления с 4px до 8px после тести-

рования с пользователями старше 55 лет, которые испытывали трудности с идентификацией кнопок». Это организационная память, которой лишены команды, держащие документацию в Notion или Confluence. Более того, файл становится контрактом между дизайном и разработкой: любое расхождение между тем, что описано в DESIGN.md, и тем, что реализовано в коде, является задокументированным багом, а не предметом интерпретации. Когда ты передаёшь этот файл в контекст AI-инструмента в начале сессии, ты передаёшь ей систему ценностей, логику принятия решений и границы допустимого, за которые нельзя выходить. Это превращает каждый последующий запрос из угадывания намерений в точное исполнение задокументированной воли.

Контрольные вопросы

В чём принципиальное отличие RAG-пайплайна от простого добавления документации в начало каждого промпта, и в каких ситуациях RAG становится необходимостью, а не просто удобством?

Почему machine-readable формат дизайн-документации должен отличаться от документации, написанной для человека, и что конкретно это означает на уровне структуры текста?

Чем DESIGN.md отличается от классического бренд-гайдлайна по аудитории, структуре и функции, и почему раздел явных запретов является в нём наиболее критичным для AI-агентов?

Как концепция Brand State Architecture меняет представление дизайнера о его роли: что именно теперь является ключевым артефактом его работы помимо самих макетов?

Глава 5: Прототип за 20 минут: от идеи до интерактивного артефакта

Умение писать хороший промпт, это необходимое условие, но не достаточное. Можно идеально сформулировать запрос, избежать всех антипаттернов, выстроить точный контекст и при этом получить на выходе красивый, но мёртвый артефакт. Потому что промпт, это инструмент передачи намерения, а не инструмент создания опыта. Опыт создаётся тогда, когда идея становится чем-то, что можно потрогать, нажать, прокрутить и почувствовать. То есть когда она превращается в прототип. Именно здесь возможности Claude Design из умозрительных становятся практически ощутимыми.

Прототипирование всегда было одной из самых энергозатратных частей дизайн-процесса, не потому что это интеллектуально сложно, а потому что между мыслью и её воплощением стоял длинный технический коридор. Figma, после правок, после согласований, после передачи во фронтенд, после интеграции. К тому моменту, когда прототип был готов к тестированию, первоначальная идея уже успевала состариться, а команда, устать от её обсуждения. AI не отменяет мышление, но убирает этот коридор. Или, по крайней мере, делает его настолько коротким, что идея доходит до тестируемого состояния, пока ещё живая.

Эта глава о том, как использовать Claude для создания прототипов, которые убеждают, проверяют гипотезы и помогают принимать решения, быстро и без потери качества суждений. Не о том, как заставить AI сделать всё за тебя. А о том, как выстроить процесс так, чтобы двадцать минут работы давали артефакт, вокруг которого можно вести осмысленный разговор с командой, заказчиком или самим собой.

8-шаговый фреймворк быстрого прототипирования с Claude

Прежде чем говорить о шагах, важно сделать одно признание: никакой фреймворк не работает как рецепт, в котором нужно точно отмерить ингредиенты. Восемь шагов, это карта маршрута. На некоторых перекрёстках ты будешь задерживаться дольше, некоторые этапы можно будет пройти за один ход. Ценность структуры не в соблюдении порядка, а в том, что она не даёт тебе перескочить с нулевой идеи сразу к пиксель-перфекту, минуя всё, что между ними.

Первый шаг, формулировка гипотезы, а не задания. Это звучит тривиально, но большинство прототипов создаются с размытой целью вроде «нужно показать онбординг». Гипотеза звучит иначе: «Если мы покажем пользователю выгоду до запроса регистрации, он с большей вероятностью дойдёт до экрана создания аккаунта». Это конкретное утверждение, которое прототип должен помочь проверить. Именно эта формулировка идёт в начало первого промпта.

Второй шаг, выбор уровня верности. Выбор между хай-фай и лоу-фай определяется тем, на что ты хочешь получить реакцию. Если тебе нужна реакция на логику потока, делай виршот с placeholder-контентом. Если тебе нужна реакция на визуальный язык бренда, делай стилизованный артефакт. Смешивать эти цели в одном прототипе, распространённая ошибка, которая приводит к тому, что люди комментируют цвета вместо флоу.

Третий шаг, описание контекста использования в промпте: кто, когда, на каком устройстве и в каком эмоциональном состоянии встречается с этим интерфейсом.

Четвёртый шаг, запрос первого артефакта с явным указанием на интерактивность: Claude должен знать, что ты хочешь HTML-прототип с рабочими переходами.

Пятый шаг, критический разбор результата: главный вопрос один — «проверяет ли это мою гипотезу».

Шестой шаг, точечная итерация с сохранением контекста: ты уточняешь конкретный элемент или сценарий, продолжая начатое.

Седьмой шаг, стресс-тест: ты намеренно проверяешь граничные случаи, длинные тексты, пустые состояния, ошибки.

Восьмой шаг, фиксация решений: короткий комментарий прямо в переписке или в отдельном документе о том, что было проверено и что решено. Этот шаг пропускают почти все, и именно поэтому через неделю никто не помнит, почему кнопка стоит слева.

Практически это выглядит так. Ты открываешь Claude, пишешь гипотезу и контекст в первом сообщении, запрашиваешь интерактивный HTML-артефакт с конкретными сценариями взаимодействия. Через минуту у тебя есть что-то работающее в браузере. Ты нажимаешь кнопки, проходишь флоу, замечаешь, что не хватает состояния загрузки на третьем экране, и пишешь об этом во втором сообщении. К третьему ходу у тебя есть прототип, который отвечает на вопрос, ради которого затевался. Достаточный для того, чтобы принять следующее решение. Смысл прототипирования состоит в создании инструмента для следующего шага мышления.

Сложные интеракции и анимации: кейс Brilliant (20+ промптов vs 2 промпта)

Есть задачи, которые кажутся принципиально неподходящими для быстрого AI-прототипирования. Сложные микровзаимодействия, образовательные интерфейсы с адаптивной логикой, анимированные визуализации данных, всё это традиционно требовало либо глубокого знания CSS-анимаций и JavaScript, либо длительной работы с Motion Design специалистом. Именно здесь разница между опытным пользователем Claude и новичком становится особенно очевидной.

Книга Brilliant, это образовательная платформа, построенная вокруг интерактивных визуальных объяснений. Когда команда хотела прототипировать новый формат интерактивного урока по теории вероятности, первый подход выглядел типично: дизайнер делал серию итераций, уточняя с каждым ходом анимацию, логику прогрессии, поведение при ошибке пользователя, состояния при правильном ответе. Двадцать с лишним промптов, разбросанных по нескольким сессиям, каждый из которых частично терял контекст предыдущего. Результат был функциональным, но процесс выматывал и не давал ощущения контроля. Второй подход появился после того, как один из дизайнеров взял паузу и выстроил промпт принципиально иначе: он описал пользовательский опыт, а именно то, что должен пережить пользователь. Эмоциональную дугу взаимодействия: момент непонимания, момент гипотезы, момент проверки, момент понимания. Плюс технические параметры: анимации через CSS transitions, состояния через классы, никаких внешних библиотек. Два промпта, и прототип, который команда смогла использовать в тестировании с реальными пользователями в тот же день.

Разница между двадцатью итерациями и двумя определяется ментальной моделью задачи. В первом случае дизайнер думал категориями элементов интерфейса: кнопка, прогресс-бар, анимация. Во втором случае он думал категориями пользовательского опыта: что человек чувствует, что он делает, что происходит в ответ. Claude значительно лучше переводит описание опыта в рабочий код, чем описание визуальных элементов в вакууме. Дизайнерам, привыкшим мыслить в терминах компонентов, такой подход кажется контринтуитивным, но как только это понимаешь, скорость работы меняется кардинально.

Для сложных анимаций работает ещё один принцип: описывай физику движения, а не параметры CSS. Вместо «добавь ease-in-out с duration 300ms» пиши «элемент появляется как будто выныривает из-под поверхности, с лёгким перелётом и возвратом». Claude переведёт это в конкретные значения кривых, и результат будет ближе к тому, что ты имел в виду, чем если бы ты указывал технические параметры напрямую, особенно если ты не анимационный дизайнер по специализации. Эта техника работает потому, что AI обучен на огромном корпусе описаний движения из кино, игрового дизайна и UX-документации, и умеет транслировать качественные метафоры в количественные значения точнее, чем многие ожидают.

Слайды, лендинги, one-pager'ы: выбор режима под задачу

Claude Design, не монофункциональный инструмент. Он одинаково хорошо создаёт интерактивные прототипы приложений, презентационные слайды, маркетинговые лендинги и компактные one-pager'ы. Но «одинаково хорошо» не означает «одинаково быстро и с одинаковым промптом». У каждого формата своя логика запроса, и смешивать их, значит получать артефакты, которые технически работают, но не решают задачу.

Слайды, это нарративный формат. Их логика подчинена последовательности аргументов, а не иерархии информации на одном экране. Когда ты просишь Claude создать слайд-презентацию, самая важная часть промпта, это описание аргументационного маршрута: с чего начинается история, какие возражения она должна снять по пути, чем заканчивается. Хороший промпт для слайдов звучит как тезисы для выступления, а не как техническое задание на дизайн. Если ты опишешь аудиторию (инвесторы серии A, которые уже видели десять похожих питчей сегодня), Claude сделает выборы в пользу лаконичности и ударных данных, а не декоративной сложности.

Лендинги, это совершенно другая задача. Здесь логика конверсионная: каждый блок должен отвечать на конкретное возражение или усиливать доверие перед следующим шагом. Промпт для лендинга должен содержать описание целевого действия (что пользователь должен сделать в итоге), описание пути к этому действию (какие сомнения нужно снять), и описание того, кто этот пользователь на уровне его внутреннего монолога в момент просмотра страницы. Claude, получив такой контекст, выстраивает структуру блоков с внутренней конверсионной логикой. Технически артефакт будет HTML с CSS, который можно открыть в браузере, прокрутить и нажать на кнопку. Для тестирования структуры и текстов этого более чем достаточно.

One-pager, самый недооценённый формат в арсенале senior-дизайнера. Это документ-убеждение, сжатый до одной страницы: идея, контекст, ключевые решения, следующий шаг. Его сила в том, что он вынуждает автора расставить приоритеты и отказаться от всего лишнего, а это само по себе форма мышления. Claude отлично создаёт one-pager'ы, если в промпте есть явное ограничение: «всё, что не умещается на одном экране без прокрутки, должно быть удалено». Это ограничение работает как дизайн-решение, и AI его соблюдает. Практически полезно просить Claude сначала написать one-pager, а потом, разместить тот же контент как лендинг: разница между двумя артефактами наглядно покажет, что теряется и что приобретается при смене формата. Это упражнение само по себе тренирует редакторское мышление.

Когда прототип достаточно хорош: критерии остановки итераций

Один из самых сложных профессиональных навыков, не умение улучшать, а умение останавливаться. В традиционном дизайне это регулировалось дедлайном и усталостью команды. В AI-прототипировании дедлайн исчезает как ограничение: следующая итерация стоит всего одного промпта и тридцати секунд ожидания. Это создаёт новую ловушку, бесконечное совершенствование артефакта, который давно перестал учить тебя чему-то новому.

Первый критерий остановки, ответ на исходную гипотезу. Если прототип создавался для проверки конкретного утверждения, как только ты можешь уверенно сказать «да» или «нет» на вопрос этой гипотезы, прототип сделал своё дело. Любые дальнейшие итерации работают уже на следующую гипотезу, и их стоит начинать с явной переформулировки цели. Это кажется очевидным, но на практике дизайнеры очень часто продолжают улучшать артефакт по инерции, не замечая, что изначальный вопрос давно закрыт и теперь они совершенствуют решение, которое ещё не было принято.

Второй критерий, стабильность реакций. Если ты показал прототип трём разным людям и получил три раза примерно одинаковый фидбек, это сигнал: паттерн зафиксирован, дальнейшие показы дадут убывающую отдачу. Если реакции принципиально расходятся, либо у тебя неоднородная аудитория (и это нужно исследовать отдельно), либо прототип не достаточно чёткий в своей идее (и это нужно исправить). В обоих случаях решение, пересмотр постановки задачи.

Третий критерий, порог передачи. Прототип достаточно хорош тогда, когда человек, получивший его без твоих пояснений, понимает идею и может дать содержательную реакцию на неё, а не на исполнение. Это хороший тест: покажи артефакт кому-то без контекста и посмотри, что именно они комментируют. Если они говорят о концепции, прототип работает. Если они говорят о том, что кнопка не того цвета или текст выглядит как placeholder, уровень верности не соответствует задаче, и это нужно либо исправить, либо явно проговорить при показе. И последнее, самое честное правило остановки: итерация нужна только тогда, когда она меняет решение, которое нужно принять сейчас. Совершенство, враг своевременности, и никакой AI не отменяет этот базовый закон продуктового дизайна.

Контрольные вопросы

Почему описание пользовательского опыта даёт Claude лучший результат, чем описание визуальных элементов, и как это использовать для сложных интерфейсов?

Как выбрать уровень верности прототипа (лоу-фай vs хай-фай) в зависимости от того, какую реакцию ты хочешь получить?

Какие конкретные детали нужно включить в промпт для лендинга, чтобы Claude выстроил блоки с конверсионной логикой, а не произвольно?

Как использовать тест «покажи без контекста» для оценки качества прототипа — и что делать, если человек комментирует исполнение вместо идеи?

Глава 6: Стили как стратегия: Swiss, Brutal, Glass, Cyber и ещё 11 — когда какой выбрать

Есть момент в каждом дизайн-процессе, когда прототип уже доказал свою состоятельность, концепция нашла подтверждение, а скелет интерфейса стоит достаточно уверенно, чтобы начать думать о коже. Именно в этот момент многие дизайнеры совершают ошибку, которую потом трудно объяснить стейкхолдерам: они начинают выбирать визуальный стиль исходя из личного вкуса, насмотренности или просто из того, что сейчас популярно в Dribbble. Это не то чтобы катастрофа, но это также и не стратегия. Это как подбирать одежду для переговоров, глядя на то, что нравится вам лично, а не на то, какое впечатление нужно произвести на другую сторону стола.

Глава о прототипировании закончилась на идее своевременности: прототип достаточно хорош тогда, когда он выполняет свою работу, а не тогда, когда он красив. Теперь мы переходим к следующему уровню сложности, где слово «красив» приобретает совершенно конкретный операциональный смысл. Потому что визуальный стиль в профессиональном дизайне, это не украшение и не финальный штрих. Это язык, на котором продукт разговаривает с пользователем ещё до того, как тот прочитал первое слово. И как любой язык, он должен соответствовать контексту, аудитории и тому, что именно ты хочешь сказать.

Почему визуальный стиль — это бизнес-решение, а не эстетика

Представь две компании, которые продают одно и то же: программное обеспечение для финансового планирования малого бизнеса. Первая оформляет свой продукт в стилистике glassmorphism с мягкими градиентами, размытыми фонами и невесомыми карточками. Вторая выбирает жёсткую швейцарскую типографику, монохромную сетку, минимум декора и максимум информационной плотности. Обе компании не ошиблись технически: оба интерфейса могут быть доступными, консистентными и хорошо реализованными. Но если первая ориентируется на фрилансеров и молодых предпринимателей, уставших от скучных корпоративных инструментов, а вторая строит продукт для бухгалтеров с двадцатилетним стажем, которые ценят точность и предсказуемость выше всего, то выбор стиля в каждом случае является таким же стратегическим решением, как ценообразование или канал дистрибуции.

Визуальный стиль сигнализирует о ценностях продукта быстрее, чем любой копирайтинг. Психологически это работает через то, что Дональд Норман описывал как visceral design: реакция происходит раньше, чем включается рациональное мышление. Пользователь видит интерфейс и за долю секунды считывает: это серьёзно или игриво, дорого или демократично, сложно или просто, для меня или не для меня. Этот первичный импульс влияет на то, даст ли человек продукту шанс вообще. И если визуальный сигнал противоречит ожиданиям аудитории, когнитивный диссонанс возникает мгновенно: грубо говоря, ты приходишь на деловую встречу в пляжных шортах. Формально ничего запрещённого, но разговора не получится.

Это означает, что выбор стиля должен начинаться не с вопроса «что нам нравится», а с трёх других вопросов. Первый: кто наш пользователь и какие визуальные коды он считывает как сигнал доверия и релевантности. Второй: какую эмоцию мы хотим создать в первые три секунды контакта, потому что именно она формирует контекст восприятия всего остального. Третий: каким должен быть продукт через три года, и не создаём ли мы стиль, который будет выглядеть как артефакт определённого момента времени. Glassmorphism, который в 2021 казался свежим, к 2024 стал восприниматься как клише, и продукты, которые сделали его основой визуального языка, теперь пожинают последствия этого решения.

Ещё один аспект, который редко обсуждают откровенно: стиль влияет на воспринимаемую ценность продукта, а значит, и на готовность платить за него. Исследования ценового восприятия в digital-продуктах показывают, что пользователи готовы платить больше за продукты, которые выглядят как premium, даже если функциональность идентична более дешёвому аналогу. Упаковка является частью продукта, и это честный разговор об устройстве вещей. Apple понимала это задолго до того, как это стало общим местом в продуктовых дискуссиях. Поэтому когда заказчик говорит тебе «просто сделай красиво», твоя работа как дизайнера состоит в том, чтобы перевести это в бизнес-язык: красиво для кого, красиво как сигнал чего и красиво в сравнении с чем из того, что уже есть на рынке.

Разбор 8 стилей ClaudeKit: сильные стороны и контексты применения

ClaudeKit систематизирует визуальный язык так, чтобы он был понятен как дизайнеру-человеку, так и языковой модели, которая будет его воспроизводить. Каждый стиль в этой системе несёт в себе определённую семантику: набор предположений о пользователе, контексте использования и эмоциональном тоне. Разберём восемь из них, которые встречаются в работе чаще всего и дают наиболее чёткую картину того, как стиль работает как инструмент.

Swiss, или International Typographic Style, основан на идее, что форма следует из содержания настолько строго, что декор становится не просто лишним, а вредным. Строгая модульная сетка, доминирование типографики над иллюстрацией, минимум цвета и максимум белого пространства. Этот стиль говорит пользователю: здесь ценится точность, информация важнее впечатления, и мы уважаем твоё время достаточно, чтобы не отвлекать от сути. Лучшие контексты: B2B-инструменты для профессионалов, финансовые продукты, медицинские приложения, любые области, где доверие строится через воспринимаемую строгость и компетентность. Худший контекст: развлекательные приложения, продукты для подростков, всё, где эмоциональный отклик важнее профессионального авторитета.

Brutal, или брутализм в цифровом пространстве, это намеренное нарушение конвенций. Видимые границы, нестандартные сетки или их полное отсутствие, сырые текстуры, грубые контрасты, ощущение, что интерфейс не старается тебе понравиться. Брутализм работает как сигнал аутентичности и антикорпоративной позиции. Он говорит: мы делаем что-то настоящее, без расчёта на всеобщее удобство. Это делает его мощным инструментом для творческих агентств, независимых медиа, культурных институций и всех продуктов, чья аудитория ценит характер выше полировки. Ошибка новичков: применять брутализм там, где он читается не как позиция, а как отсутствие бюджета. Разница между намеренным хаосом и просто небрежным дизайном заметна опытному глазу сразу.

Glass, или glassmorphism, построен на метафоре прозрачности и глубины: размытые фоны, полупрозрачные поверхности, мягкие тени, ощущение многослойного пространства. Этот стиль хорошо работает там, где нужно создать ощущение современности, лёгкости и технологичности без агрессии. Его слабость, стилистическая датированность: glassmorphism настолько плотно ассоциируется с конкретным периодом в дизайне, что требует очень умелого применения, чтобы не выглядеть как артефакт. Хорошие контексты: потребительские приложения с сильным эмоциональным компонентом, health и wellness, финтех для молодой аудитории. Cyber, или киберпанк-эстетика, работает на контрасте тёмных фонов, неоновых акцентов, технологических текстур и ощущения будущего с привкусом тревоги. Это сильный стиль для игровых продуктов, криптопространства, security-инструментов и всего, что апеллирует к субкультуре технологического андеграунда. Проблема в том, что он требует очень точного дозирования: перебор превращает интерфейс в костюм для Хэллоуина.

Material Design в его современном воплощении, это система, в которой физические метафоры смягчились до почти неощутимых, но принципы остались: поверхности, тени, движение как носитель смысла. Это стиль по умолчанию для экосистемы Android и Google-продуктов, и его сила, в зрелости и отработанности. Flat 2.0, его более минималистичный родственник, убирает даже минимальные намёки на физику и работает через цвет и типографику. Хорошо подходит для продуктовых дизайнеров, работающих внутри зрелых экосистем, где консистентность с окружающей средой важнее оригинальности.

Neumorphism, который несколько лет назад казался откровением, сегодня является едва ли не учебниковым примером стиля, у которого проблема с accessibility встроена в саму его

ДНК: мягкие тени и слабые контрасты делают элементы плохо различимыми для людей с нарушениями зрения, и никакой эстетической привлекательностью это не компенсируется.

Editorial style, наконец, это дизайн, вдохновлённый журнальной и книжной вёрсткой: крупная типографика, асимметричные сетки, смелые текстовые блоки как визуальные элементы. Он работает для медиапродуктов, онлайн-изданий, портфолио и лендингов, где контент сам по себе является главным героем.

Миксинг стилей: как создать уникальный визуальный язык через AI

Самая большая ловушка в работе с готовыми стилевыми категориями состоит в том, что они кажутся взаимоисключающими. Swiss или Brutal. Glass или Flat. Но реальность дизайна богаче любой таксономии: все продукты, которые обладают по-настоящему узнаваемым визуальным языком, являются в той или иной степени гибридами. Notion соединяет editorial-типографику с минимализмом, близким к Swiss, и добавляет к этому почти сырую, текстовую ощущаемость, которая перекликается с брутализмом. Linear берёт технологическую эстетику с тёмным фоном и неоновыми акцентами и дисциплинирует её строгой сеткой и швейцарской точностью в типографике. Гибридность результата делает визуальный язык продукта запоминаемым.

Когда ты работаешь с Claude Design в режиме миксинга стилей, ключевым навыком становится умение формулировать не просто список референсов, а логику соединения. Это означает, что промпт должен объяснять не только что ты берёшь из каждого стиля, но и почему именно эти элементы, и как они должны сосуществовать. Например: «Возьми строгость сетки и иерархию типографики из Swiss International Style, но замени нейтральную цветовую палитру на глубокие насыщенные тона с одним ярким акцентным цветом, характерным для Editorial. Поверхности карточек должны иметь очень слабое, почти неощутимое frosted glass качество, но только как фоновый эффект, не как главный визуальный приём. Общее ощущение должно читаться как serious but alive, инструмент для профессионала, который не хочет работать в скучной среде». Это совершенно другой уровень инструкции по сравнению с «сделай в стиле Swiss с небольшими акцентами».

Практически миксинг стилей с AI лучше всего работает через итерационное раскрытие. Начни с доминирующего стиля и получи базовый артефакт. Затем описывай отклонения от него последовательно, по одному: измени то, добавь это, убери вот это. Этот подход позволяет тебе контролировать каждый шаг трансформации и понимать, что именно изменилось. Один сложный промпт со всеми параметрами сразу может дать интересный результат, но без понимания того, какие именно элементы создали нужный эффект, воспроизвести его в следующей сессии будет сложнее. Итерационная логика миксинга, это ещё и документация: каждый шаг становится частью истории визуального языка, которую ты можешь показать команде или клиенту.

Отдельного разговора заслуживают «стилистические противоречия», которые при правильном управлении создают самые запоминающиеся визуальные языки. Взять жёсткость брутализма и смягчить её через цвет и закруглённые углы, это звучит как оксюморон, но именно эта формула лежит в основе визуального языка ряда успешных продуктовых компаний, которые хотят казаться одновременно бескомпромиссными и дружелюбными. Попросить AI воспроизвести это противоречие в лоб не получится: нужно описывать результирующее ощущение, а не процедуру его достижения. «Интерфейс должен выглядеть как будто его создал очень хорошо одетый анархист», это смешно звучит, но как инструкция для AI работает лучше, чем «смешай Swiss и Brutal». Характер первой формулировки AI воспроизводит лучше, чем технические категории.

Тест на вкус: как обучить AI вашему личному стилевому чутью

Вот вопрос, который не звучит в большинстве руководств по промптингу, но является, пожалуй, самым важным для дизайнера, который хочет использовать AI как подлинный творческий инструмент, а не как умную систему автозаполнения: как сделать так, чтобы AI понимал твой вкус, а не просто выполнял твои инструкции? Разница между этими двумя состояниями примерно такая же, как между ассистентом, который исполняет буквальные команды, и ассистентом, который понимает твои намерения достаточно глубоко, чтобы предвидеть, что именно ты имеешь в виду, когда говоришь «это недостаточно напряжённо» или «здесь не хватает воздуха».

Первый шаг к обучению AI твоему стилевому чутью, это создание того, что можно назвать вкусовым словарём. Это набор прилагательных и метафор, которые ты используешь, когда описываешь дизайн, который тебе нравится, и объяснений того, что именно за этими словами стоит. «Напряжённый» для тебя может означать высокий контраст и сжатые поля, а для кого-то другого, тёмную цветовую палитру и угловатые формы. Эта разница критична, и её нужно проговорить явно. Создай документ, где ты берёшь десять прилагательных, которые чаще всего используешь в дизайн-критике, и расшифровываешь каждое из них через конкретные визуальные параметры: цвет, типографику, отступы, формы, плотность информации, характер анимации. Этот документ становится частью твоего системного промпта или контекста, который ты даёшь в начале каждой сессии.

Второй инструмент, это метод сравнительной калибровки. Он работает следующим образом: ты даёшь Claude два референса, которые ты сам считаешь полярными по качеству или подходу, и описываешь, почему один из них тебе ближе. Просто сказать «мне больше нравится этот» недостаточно: важно объяснить, как в этом варианте есть что-то, что я бы назвал достоинством: типографика присутствует с весом и уверенностью. Отступы работают как пауза в хорошей речи, а не как незаполненное пространство. Чем больше таких сравнений ты проводишь в рамках одной сессии, тем точнее модель калибрует то, что ты имеешь в виду под «хорошим». Это аналогично тому, как работает рейтинговая система в рекомендательных алгоритмах: каждое сравнение сужает пространство неопределённости.

Третий, и, пожалуй, наиболее недооценённый метод: обучение через отказ. Когда AI предлагает вариант, который тебе не нравится, не просто проси переделать. Формулируй, что именно не работает и почему, даже если это сложно articulate в первый момент. «Это слишком старается понравиться. Ощущение, что дизайн тревожится о том, понравится ли он мне, и именно эта тревожность мне мешает», это звучит почти как литературная критика, но это именно тот тип обратной связи, который обучает модель твоей системе ценностей быстрее, чем технические правки. Совокупность таких отказов с объяснениями постепенно строит в рамках сессии нечто вроде негативного пространства твоего вкуса: модель начинает понимать, чего ты точно не хочешь, и это уже является мощным ограничением для генерации. Помни, что этот калибровочный процесс живёт в рамках одной сессии, если у тебя нет системы сохранения контекста. Поэтому инвестиция времени в подготовку хорошего стартового документа с твоим вкусовым словарём окупается каждый раз, когда ты открываешь новый проект.

Контрольные вопросы

Как обосновать выбор визуального стиля перед клиентом, который настаивает на своих эстетических предпочтениях, противоречащих потребностям целевой аудитории?

Почему neumorphism оказался нежизнеспособным с точки зрения accessibility и какие уроки из этого можно извлечь для оценки новых стилевых трендов?

Что такое вкусовой словарь дизайнера и как его правильно составить, чтобы он работал как эффективная инструкция для языковой модели?

Почему описание характера и эмоции интерфейса в промпте работает лучше, чем прямое перечисление технических стилевых категорий?

Глава 7: Claude Code как суперсила дизайнера: передача макета в production без потерь

Стили как стратегия закрыли важный вопрос: что именно генерировать и почему именно так. Но есть следующий вопрос, который опытный дизайнер задаёт сразу после того, как прототип доказал свою состоятельность. Он звучит не как «хорошо ли это выглядит?» и не как «понравится ли это пользователю?». Он звучит жёстче и практичнее: «Как это попадёт в продакшн без потерь?» Именно здесь большинство дизайн-процессов спотыкается. Не на этапе идеи, не на этапе прототипа и даже не на этапе утверждения у стейкхолдеров. На переходе от Figma к коду.

Этот переход традиционно называли хэндоффом, и само слово уже содержит в себе проблему. Передача из рук в руки предполагает, что что-то меняет владельца. И вместе со сменой владельца неизбежно теряется часть смысла. Разработчик получает макет, но не получает контекст решений. Он видит, что кнопка круглая, но не знает, почему она круглая, что произошло бы, если бы она была прямоугольной, и какое пользовательское исследование за этим стоит. Зазор между замыслом и реализацией существует не потому, что разработчики плохо читают макеты, и не потому, что дизайнеры плохо их оформляют. Он существует потому, что традиционный хэндофф передаёт результат, но не передаёт мышление.

Claude Code меняет эту механику принципиально. Не потому, что он «лучше переводит Figma в код», хотя это тоже правда. А потому, что он способен работать с дизайн-интентом как с полноценным типом данных. Эта глава о том, как именно это работает, почему разрыв между дизайном и реализацией имеет структурные причины, которые не исчезнут сами по себе, и как выстроить пайплайн, в котором твой замысел добирается до продакшна в целости.

Разрыв между дизайном и реализацией: почему он существует и как AI его закрывает

Чтобы понять природу разрыва, нужно честно посмотреть на то, что именно передаётся в момент хэндоффа. Дизайнер создаёт артефакт, который содержит визуальные решения, но почти никогда не содержит систему аргументов, которая за ними стоит. Zeplin, Figma Dev Mode, Avocode и любой другой инструмент хэндоффа исторически решали одну задачу: точнее передать пиксели. Отступы, цвета, типографику, размеры. Это полезно, но это трансляция формы без трансляции смысла. Разработчик получает ответ на вопрос «что», но редко получает ответ на вопрос «почему». А именно «почему» определяет, как правильно обработать пограничные случаи, как масштабировать компонент, как поступить с состоянием, которое дизайнер не предусмотрел в макете.

Есть ещё одна структурная проблема, о которой говорят меньше. Дизайнер и разработчик работают в разных системах координат, и это буквальное описание реальности. Дизайнер думает в категориях пользовательского опыта, визуальной иерархии, состояний и переходов. Разработчик думает в категориях компонентного дерева, пропсов, эффектов и производительности. Эти системы координат пересекаются, но не совпадают. И каждый раз, когда они не совпадают, возникает интерпретационный люфт. Разработчик принимает решение самостоятельно, потому что другого варианта нет. Иногда это решение совпадает с замыслом дизайнера. Чаще не совпадает ровно настолько, чтобы накопиться в продакте в виде едва заметных, но раздражающих несоответствий.

AI закрывает этот разрыв благодаря появлению общего посредника, который понимает оба языка. Claude Code способен принять дизайн-решение, описанное в терминах UX и визуального замысла, и транслировать его в код с сохранением логики, а не только геометрии. Он может объяснить разработчику, почему компонент устроен именно так, какие состояния являются критическими для пользователя, а какие можно упростить без ущерба для опыта. Посредник, снижающий стоимость коммуникации между двумя профессиональными культурами, исторически говорившими мимо друг друга, человеческое суждение при этом остаётся незаменимым.

Важно понимать, что разрыв никуда не исчезает сам по себе от того, что у тебя есть Claude Code. Он закрывается только тогда, когда дизайнер сознательно перестраивает свой процесс под новую механику. Это требует нового навыка: делать хороший дизайн и документировать его так, чтобы он оставался хорошим дизайном после перехода в чужие руки и другую среду. Именно об этом следующий раздел.

Хэндофф нового поколения: от Claude Design к Claude Code за один пайплайн

Традиционный хэндофф выглядел как эстафета: дизайнер заканчивал свой этап, передавал палочку разработчику и в лучшем случае оставался доступен для вопросов. В худшем случае дизайнер уходил на следующий спринт, а разработчик разбирался с макетом самостоятельно. Хэндофф нового поколения устроен принципиально иначе: это непрерывная цепочка, где один и тот же контекст путешествует от концепции до компонента, не теряя смысла на каждом переходе.

Практически это выглядит так. Когда ты работаешь с Claude Design и создаёшь прототип или компонент, у тебя уже есть история взаимодействия: промпты, которые ты писал, артефакты, которые получились, итерации и решения, которые ты принял в процессе. Этот контекст можно передать в Claude Code напрямую как стартовую точку, минуя ручной перевод в спецификации. Claude Code получает не только финальный визуальный результат, но и логику, которая к нему привела. Это принципиально меняет качество генерируемого кода: вместо кода, который просто воспроизводит внешний вид, ты получаешь код, который воспроизводит намерение.

Пайплайн в своём базовом виде состоит из нескольких шагов, которые важно выстроить последовательно. На первом этапе ты работаешь в Claude Design и параллельно ведёшь что-то вроде дизайн-журнала прямо внутри сессии: фиксируешь ключевые решения, объясняешь, почему ты выбрал именно этот паттерн, а не соседний, какие альтернативы были отброшены и по какой причине. На втором этапе этот журнал вместе с финальным артефактом передаётся в Claude Code как контекст. На третьем этапе Claude Code генерирует не просто разметку и стили, а компонент с встроенными комментариями, которые объясняют архитектурные решения в терминах, понятных разработчику. Это звучит как дополнительная работа, но на практике занимает меньше времени, чем последующие раунды вопросов и правок после стандартного хэндоффа.

Есть один нюанс, который опытные дизайнеры замечают довольно быстро. Пайплайн работает тем лучше, чем чище структурирован контекст, который в него подаётся. Если твоя сессия в Claude Design была хаотичной: много экспериментов, множество изменений направления без объяснений, итоговый артефакт, который возник как бы ниоткуда, то Claude Code будет реконструировать логику решений по косвенным признакам. Это лучше, чем ничего, но хуже, чем мог бы дать аккуратно структурированный контекст. Хороший пайплайн начинается уже в момент, когда ты открываешь первый промпт.

Дизайн-интент в коде: как упаковать «почему» вместе с «что»

Дизайн-интент это термин, который в индустрии используют неточно. Чаще всего под ним понимают «то, что имел в виду дизайнер», как если бы это была некая тайна, которую нужно раскрыть. На самом деле дизайн-интент это структурированное описание решения, включающее четыре элемента: что именно сделано, почему именно так, какие пограничные случаи покрыты и какие намеренно оставлены за скобками, и как это решение должно вести себя при изменении контекста. Когда все четыре элемента закодированы в передаваемом артефакте, хэндофф перестаёт быть источником потерь.

Claude Code позволяет упаковать дизайн-интент в код несколькими способами. Первый и самый очевидный, это комментарии, но содержательные, объясняющие решение в контексте пользовательского опыта, а не беспомощные однострочники вида «кнопка основного действия». Например: «Отступ между иконкой и лейблом составляет восемь пикселей, а не шесть, потому что в юзабилити-тесте пользователи с низким зрением не могли уверенно идентифицировать связь между элементами при меньшем расстоянии». Такой комментарий не просто описывает значение токена, он передаёт аргумент, который стоит за этим значением. Разработчик, который столкнётся с необходимостью изменить этот отступ, будет принимать решение осознанно, а не на глаз.

Второй способ тонкий, но мощный: именование переменных и пропсов как дизайн-решений. Когда компонент принимает пропс под названием «isDestructive» вместо «isRed», он уже несёт в себе семантику замысла. Разработчик понимает, что красный цвет здесь выполняет функциональную роль: сигнализирует об опасности действия, а не выражает эстетическое предпочтение. Claude Code при грамотном промптинге генерирует именно такие структуры, потому что ты можешь напрямую инструктировать его именовать элементы в терминах UX-намерений, а не визуальных атрибутов. Это маленькое изменение, которое накапливается в большую разницу на уровне кодовой базы.

Третий способ, который большинство дизайнеров ещё не освоили, это документирование состояний как первоклассных граждан компонента. Традиционный хэндофф передаёт дефолтное состояние и иногда hover. Всё остальное разработчик додумывает. Дизайн-интент в коде, сгенерированный через Claude Code с правильным контекстом, включает все намеренно спроектированные состояния: disabled, loading, error, success, empty, partial, и для каждого объясняет, какое пользовательское действие или системное событие его вызывает. Задача здесь одна: правильно сформулированный промпт поручает Claude Code самому вывести и задокументировать эти состояния на основе предоставленного контекста.

Кейс Datadog: от прототипа до production-компонента

Datadog, платформа мониторинга инфраструктуры, работает в контексте, где дизайн-система несёт на себе экстремальную нагрузку. Продукт используется инженерами в момент инцидента: когда что-то горит, когда каждая секунда стоит денег и когда когнитивная нагрузка и без того максимальная. В таких условиях зазор между замыслом дизайнера и реализацией разработчика не просто эстетическая проблема. Это проблема операционной безопасности. Если компонент алерта ведёт себя не так, как ожидает инженер во время инцидента, последствия выходят за пределы плохого UX.

Команда дизайна Datadog столкнулась с конкретной задачей: нужно было спроектировать и реализовать компонент системного алерта, который корректно работал бы в четырёх контекстах размещения, поддерживал семь типов критичности, адаптировался к тёмной и светлой теме и при этом оставался производительным в условиях потока из сотен одновременных событий. Традиционный путь: дизайнер делает макеты всех состояний, пишет спецификацию, передаёт разработчику, разработчик задаёт двадцать вопросов, дизайнер отвечает в течение двух дней, разработчик реализует, дизайнер проверяет и находит расхождения. Итого около двух недель только на один компонент.

С пайплайном Claude Code этот цикл сжался до принципиально иного масштаба. Дизайнер создал интерактивный прототип компонента в Claude Design, намеренно ведя диалог так, чтобы каждое ключевое решение было зафиксировано в тексте сессии. Почему именно такая иерархия типографики для критических алертов. Почему анимация появления ограничена пятью сотыми секунды, а не стандартными двумястами миллисекундами. Почему иконка критичности вынесена левее лейбла, а не помещена рядом с ним. Этот контекст вместе с прототипом был передан в Claude Code с инструкцией: «Сгенерируй React-компонент, который точно воспроизводит этот прототип, включая все задокументированные состояния, и упакуй дизайн-решения в комментарии на уровне кода таким образом, чтобы разработчик, который никогда не видел этот дизайн, мог понять обоснование каждого нестандартного решения».

Результат первой итерации закрыл около восьмидесяти пяти процентов требований. Оставшиеся пятнадцать потребовали двух дополнительных промптов: один касался обработки состояния, когда алерт появляется одновременно с другим алертом той же критичности, второй уточнял поведение в условиях очень длинного текста сообщения. Общее время от прототипа до production-ready компонента составило около четырёх часов вместо двух недель. Но важнее временного показателя другое: разработчик, принявший компонент, не задал ни одного вопроса о замысле. Все вопросы, которые возникают при традиционном хэндоффе, уже были отвечены внутри кода. Хэндофф нового поколения строится на умной упаковке, в которой замысел путешествует вместе с реализацией и прибывает на место назначения в целости.

Контрольные вопросы

Что конкретно отличает хэндофф через Claude Code от стандартной передачи спецификаций разработчику?

Почему именование пропсов в терминах UX-намерений, а не визуальных атрибутов, влияет на качество кодовой базы в долгосрочной перспективе?

Какой навык дизайнера становится ключевым в условиях AI-ассистированного хэндоффа и почему он требует переосмысления привычного процесса?

Глава 8: 18 Claude Code Skills для UX/UI: полный разбор и рейтинг по ROI

Кейс Datadog показал кое-что важное помимо самого кейса. Он показал, что разрыв между дизайном и реализацией закрывается не тогда, когда дизайнер учится писать код, и не тогда, когда разработчик учится думать о пользователях. Он закрывается, когда обе стороны начинают работать через общий язык, который понимают все участники процесса. Claude Code в этой истории выступил не переводчиком, а медиатором, и его эффективность напрямую зависела от того, насколько точно дизайнер умел активировать нужные режимы его работы.

Это подводит нас к разговору, который давно нужно было провести честно и системно. Claude Code, это не монолитный инструмент с одной большой кнопкой «сделай хорошо». Это среда с восемнадцатью отдельными навыковыми режимами, каждый из которых заточен под конкретный тип задачи и даёт разную отдачу в зависимости от контекста. Большинство дизайнеров, начинающих работать с Claude Code, используют от двух до четырёх из них, интуитивно нащупывая те, что дают видимый результат. Это не катастрофа, но это примерно как пользоваться Figma, зная только инструмент прямоугольника и экспорт в PNG. Ты формально работаешь в Figma, но твой потолок искусственно занижен.

Эта глава, попытка дать полную карту. Не теоретический список возможностей из документации, а практический разбор с точки зрения ROI: что даёт максимальную отдачу при минимальных вложениях времени, что требует инвестиций, но окупается на дистанции, а что можно смело игнорировать, если твоя специализация не предполагает конкретных сценариев. И, что важно, как эти режимы взаимодействуют между собой, потому что настоящая суперсила начинается не когда ты освоил каждый из них по отдельности, а когда научился их комбинировать.

Frontend Design, Interface, Taste: топ-5 скиллов с максимальной отдачей

Если бы нужно было выбрать пять навыковых режимов Claude Code, которые дают наибольшую отдачу именно для UX/UI-дизайнера, а не для fullstack-разработчика или продакта, список получился бы неожиданно конкретным. Эти пять находятся на пересечении того, что дизайнер умеет формулировать лучше всего, и того, что Claude Code умеет реализовывать с наименьшим количеством потерь при трансляции.

Первый и, пожалуй, наиболее очевидный, это Frontend Design. Режим, в котором Claude Code работает с визуальной структурой интерфейса: раскладкой, типографикой, системой отступов, иерархией. Это возможность передать семантику макета на языке, который одновременно понимают и дизайнер, и компилятор. Ключевое отличие от просто генерации кода состоит в том, что в режиме Frontend Design Claude Code сохраняет дизайн-интент на уровне структуры, а не только на уровне пикселей. Грубо говоря, он воспроизводит компонент с пониманием причин его визуального решения и способен адаптировать это «почему» при изменении условий контейнера или брейкпоинта.

Второй скилл, Interface, работает иначе. Если Frontend Design, это про визуальную структуру, то Interface, это про поведение. Состояния компонентов, переходы между режимами, микроинтеракции, реакция на пользовательский ввод. Именно здесь большинство дизайнеров исторически теряли больше всего при хэндоффе: разработчик видел статичный макет, не видел hover-состояния при зажатом клике, не знал, как именно должна выглядеть ошибка валидации после потери фокуса, и принимал решения самостоятельно. Interface как режим позволяет упаковать все эти состояния в один связный артефакт, который не требует отдельного documentation-митинга для объяснений.

Третий, и самый интересный с точки зрения дизайн-философии, это Taste. Режим, который поначалу кажется наименее операциональным, но на практике оказывается одним из наиболее мощных. Taste позволяет Claude Code оценивать и корректировать визуальные решения с точки зрения качества, а не только технической корректности. Когда ты передаёшь ему компонент и просишь «сделай это лучше», он применяет принципы визуальной иерархии, типографического ритма, консистентности с окружающим контекстом. Taste служит масштабированием дизайнерского вкуса: ты задаёшь критерии качества один раз, а потом Taste режим применяет их автономно к большому числу компонентов.

Четвёртый скилл по ROI, это Accessibility. В смысле автоматической проверки контрастности, и сверх того. В смысле способности генерировать семантически корректную разметку, ARIA-атрибуты, управление фокусом и клавиатурную навигацию как часть первоначального артефакта, а не как отдельный аудит после. Для дизайнера, который работает с продуктами в публичном секторе, финтехе или медицине, где стандарт WCAG 2.1 AA является де-факто обязательным требованием, экономия от этого скилла измеряется не часами, а целыми циклами ревью. Пятый скилл, Responsive Behavior, заслуживает отдельного внимания: адаптивная вёрстка с брейкпоинтами здесь опирается на понимание того, как информационная иерархия должна меняться при изменении контейнера. Разница между разработчиком, который скрывает колонки ниже 768 пикселей, и режимом Responsive Behavior в том, что последний понимает, что скрывать нужно вторичный контент, а не тот, что находится правее по макету.

Скиллы для системной работы: масштабирование компонентов и токенов

Отдельного разговора заслуживает группа скиллов, которая начинает давать отдачу не сразу, а с определённого порога сложности. Если ты работаешь над одним продуктом с дизайн-системой из сорока компонентов, некоторые из этих инструментов покажутся избыточными. Но как только система переваливает за сотню компонентов, а команда разрастается до трёх и более дизайнеров, эти скиллы превращаются из бонуса в инфраструктуру.

Token Management, первый из системных скиллов, решает задачу, которую принято недооценивать до тех пор, пока она не превращается в технический долг с шестизначной стоимостью исправления. Речь о консистентности дизайн-токенов между Figma, кодовой базой и документацией. Claude Code в режиме Token Management умеет работать с токенами как с типизированными данными: проверять соответствие именований, детектировать дублирование семантически схожих значений, предлагать реструктуризацию при масштабировании системы. Для дизайнера, который раньше делал это вручную через сравнение JSON-файлов и скриншотов Figma-переменных, разница в скорости работы измеряется порядком величины.

Component Generation работает в связке с Token Management и решает следующую по сложности задачу: как создавать новые компоненты, которые органично вписываются в существующую систему, а не нарушают её. Классическая проблема роста дизайн-системы состоит в том, что каждый новый компонент, созданный под давлением дедлайна, немного отклоняется от системного паттерна. Через год у тебя шесть вариантов отображения дат, три разные иконки «закрыть» и четыре реализации тултипа, ни одна из которых не совпадает с остальными. Component Generation в режиме Claude Code позволяет задавать новый компонент через описание его функции и контекста, а не через ручное копирование существующих паттернов. Система сама применяет правила неймингов, наследует токены и соблюдает структурные соглашения.

Documentation Generation часто воспринимается как наименее гламурный из системных скиллов, но на практике именно он закрывает одну из наиболее болезненных точек командной работы. Дизайн-система без внятной документации, это дорогой библиотечный каталог, которым никто не умеет пользоваться. Claude Code в режиме Documentation Generation умеет извлекать из компонентного кода пропсы, вариации, правила использования и ограничения, а затем форматировать это в удобочитаемую документацию, которую можно читать и разработчику, и дизайнеру, и продакту. Важный нюанс: технический слой документации поддерживается актуальным без отдельного ресурса на его обновление, тогда как содержательная документация по-прежнему требует человека.

Version Diff, последний из скиллов, которые я бы отнёс к системной группе с высоким ROI, решает задачу, появившуюся именно в контексте AI-ускоренного дизайна. Когда компоненты генерируются и итерируются быстро, становится критически важным понимать, что именно изменилось между версиями и почему. Version Diff позволяет Claude Code анализировать два состояния компонента или токена и производить структурированный changelog с разбивкой по типам изменений: визуальные, поведенческие, структурные. Для дизайнера, который ведёт систему в команде с историей принятия решений, это эквивалент ADR-документов в инженерной культуре, только автоматически генерируемый из самих артефактов.

Как комбинировать скиллы для нестандартных задач

Вот где начинается настоящая территория продвинутой работы с Claude Code. Каждый скилл в отдельности решает конкретную задачу. Но дизайн редко приходит в виде конкретных задач. Он приходит в виде размытых проблем, у которых нет очевидного решения, и именно в этих случаях умение комбинировать режимы отличает специалиста, работающего эффективно, от того, кто работает быстро.

Рассмотрим конкретный сценарий, который большинство дизайнеров встречает хотя бы раз в квартал: редизайн существующего компонента, который уже живёт в production и используется в пятнадцати разных контекстах. Задача в том, чтобы обновить его визуально, не сломав ничего из уже работающего поведения и не создав регрессии в тех местах, где его применяли с небольшими вариациями. Попытка решить это одним скиллом обречена на провал. Frontend Design покажет, как он должен выглядеть, но не учтёт поведенческих edge cases. Interface покроет поведение в нормальных сценариях, но пропустит то, как компонент ведёт себя при динамических данных. Version Diff зафиксирует изменения, но не оценит их целостность. Правильный подход, последовательная цепочка: сначала Frontend Design для формирования визуального решения, затем Interface для описания поведенческой спецификации, затем Accessibility для проверки семантики в новом контексте, и Version Diff на финале для документирования решения с обоснованием каждого изменения.

Другой распространённый сценарий, создание нового продукта с нуля в условиях жёсткого временного ограничения. Здесь соблазн использовать только Taste и Frontend Design велик: они дают быстрый красивый результат. Но если параллельно не активировать Token Management, через неделю у тебя будет визуально привлекательный интерфейс, который невозможно масштабировать, потому что в нём нет системности ниже поверхности. Комбинация, которая работает в этом контексте, выглядит иначе: Taste для определения визуального языка, Token Management для его немедленной систематизации, Component Generation для первых компонентов, и Responsive Behavior для проверки устойчивости системы на разных устройствах с первой же итерации.

Есть ещё один тип комбинирования, который не очевиден без практики. Это использование скиллов в «следящем» режиме, когда один режим работает как валидатор для другого. Например, после каждой итерации Frontend Design можно запускать Accessibility в режиме проверки: цель одна — немедленный сигнал о регрессии в семантике. Это создаёт петлю обратной связи внутри самого инструмента, которая позволяет двигаться быстро, не накапливая технический долг по доступности. Именно такая схема описывалась в кейсе Datadog: там пайплайн не просто генерировал компонент, он одновременно валидировал его через несколько линз качества, и это позволило сократить цикл ревью от нескольких спринтов до двух дней.

Отдельно стоит поговорить о комбинациях, которые не работают. Попытка одновременно запустить Taste и Token Management на раннем этапе, когда токеновая структура ещё не определена, создаёт конфликт: Taste оптимизирует под визуальный результат, Token Management требует структурной строгости, и эти два вектора тянут артефакт в разные стороны. Аналогично, совместное использование Component Generation и Documentation Generation на этапе прототипа, это преждевременная оптимизация, которая создаёт иллюзию зрелости системы раньше, чем система действительно созрела. Хороший эмпирический признак готовности к комбинированию: если ты можешь сформулировать ограничения каждого из включаемых режимов и понимаешь, почему именно их сочетание решает задачу лучше, чем любой из них в отдельности, ты готов к этому шагу.

Строим собственный скилл-стек под свою специализацию

Концепция универсального скилл-стека для дизайнера, это миф, причём вредный. Продуктовый дизайнер, работающий над B2B SaaS-платформой, сталкивается с совершенно другим набором задач, чем дизайнер систем в e-commerce или UX-исследователь в медтехе. Попытка освоить все восемнадцать скиллов равномерно, это стратегия разведчика там, где нужна стратегия снайпера. Ты везде хорошо, нигде отлично, и ни в одной точке у тебя нет конкурентного преимущества.

Начать стоит с честного аудита своей специализации через призму трёх вопросов. Первый: где ты проводишь большую часть времени прямо сейчас? Важно твоё реальное местонахождение, а не желаемое. Если ответ «в Figma, итерируя компоненты дизайн-системы», твой приоритетный стек, это Token Management, Component Generation и Version Diff с поддерживающей ролью Frontend Design. Второй вопрос: что является твоим самым болезненным узким местом? Место, где ты теряешь больше всего времени или где качество результата тебя систематически расстраивает. Если это передача дизайна в production, скилл-стек строится вокруг Interface и Accessibility как основного содержания хэндоффа. Третий вопрос: что тебе предстоит делать больше через год? Если ты видишь движение в сторону более старшей роли с фокусом на стратегию и системы, Documentation Generation и Version Diff становятся инвестицией в будущую должность, а не просто инструментами текущей работы.

Для продуктового дизайнера, работающего в формате «от исследования до прототипа», базовый стек выглядит так: Taste как основной инструмент визуального принятия решений, Interface для спецификации поведения, Responsive Behavior для быстрой проверки устойчивости решений, и Accessibility как обязательный проверочный слой. Это четыре скилла, которые покрывают восемьдесят процентов реальных задач в этой специализации. Остальные четырнадцать остаются в арсенале, но не требуют глубокого освоения прямо сейчас.

Для дизайнера систем, отвечающего за развитие компонентной библиотеки, стек переворачивается почти зеркально. Token Management и Component Generation становятся ядром, Documentation Generation, постоянным спутником, Version Diff, инструментом еженедельного применения. Frontend Design и Taste смещаются на периферию: они нужны при создании принципиально новых компонентов, но не при масштабировании существующих паттернов. Для фрилансера, который делает объём небольшого агентства в одиночку, логика сборки стека другая: ищешь максимальную ширину покрытия при минимальном количестве осваиваемых режимов. Здесь Frontend Design, Taste и Responsive Behavior дают наибольший охват, потому что они работают на всём спектре клиентских задач от лендинга до MVP.

Один из самых полезных практических шагов при сборке персонального стека, это ведение короткого журнала задач в течение двух недель. Просто фиксируй, чем занимался каждый день с точностью до типа работы. Через две недели у тебя будет эмпирическая картина того, где реально расходуется твоё профессиональное время, и она почти наверняка будет сильно отличаться от того, каким ты видишь свою работу в теории. Эта картина и есть правильное основание для сборки скилл-стека. Основание строится на реальных задачах вторника, среды и четверга, а не на категории «дизайнер систем» в строке должности. Claude Code, как и любой инструмент с высоким потолком возможностей, наиболее эффективен не тогда, когда освоен полностью, а тогда, когда освоен точно под конкретный контекст его применения.

Контрольные вопросы

Как определить, какие из восемнадцати скиллов Claude Code актуальны именно для вашей специализации, не тратя месяцы на проверку каждого из них?

Как скилл Interface меняет формат хэндоффа между дизайном и разработкой, и какие конкретно типы потерь при передаче он устраняет?

Как выглядит персональный аудит задач, который помогает собрать скилл-стек на основе реальной практики, а не теоретических представлений о своей роли?

Персональный скилл-стек, собранный под реальные задачи вторника, среды и четверга, это хорошая точка опоры. Но стек, каким бы точно настроенным он ни был, работает лишь настолько эффективно, насколько богат материал, который в него поступает. И здесь начинается разговор, который в большинстве руководств по AI-дизайну почему-то стыдливо замалчивают: о том, что именно ты даёшь модели на входе. Потому что даже самый точный скилл-стек и самый выверенный промпт не компенсируют бедности контекста. Мусор на входе, мусор на выходе, как говорили инженеры данных задолго до появления генеративных моделей, и этот принцип никуда не делся.

Мультимодальность изменила правила игры именно потому, что она сняла вопрос о формате входных данных как ограничении. До определённого момента работа с AI предполагала обязательную предобработку: нужно было переводить всё в текст, вручную извлекать суть из брифа, пересказывать структуру экрана словами, транскрибировать паттерны конкурента. Это была невидимая работа, которую никто не считал работой, но которая съедала реальное время и неизбежно вносила искажения, потому что любой пересказ это уже интерпретация. Сегодня эта прослойка уходит. Бриф в DOCX, Figma-файл, скриншот конкурентского лендинга, фрагмент кодовой базы, всё это становится прямым входом, не требующим ручной трансляции. И именно здесь находится один из самых реальных источников конкурентного преимущества для дизайнера, который понимает, как этим пользоваться.

Импорт реального мира: как скормить AI бриф в любом формате

Реальный рабочий процесс дизайнера выглядит иначе, чем его описывают в учебниках. Он начинается с письма в мессенджере, голосового сообщения от менеджера продукта, PDF-презентации, которую маркетинг собрал из трёх разных источников, Excel-таблицы с результатами опроса пользователей и Word-документа, в котором юристы внесли правки поверх правок. Всё это нужно как-то переварить, вычленить суть и превратить в рабочую постановку задачи. Традиционно этот этап занимал от нескольких часов до нескольких дней и целиком лежал на плечах дизайнера. Мультимодальный ввод позволяет переложить значительную его часть на модель, но только если понимать, как именно это делать.

Самый прямой способ работы с документами в Claude, это загрузка файла как части контекста. DOCX, PDF, CSV, TXT, изображения в любом распространённом формате, всё это Claude обрабатывает нативно. Принципиально важна, однако, техническая возможность загрузки в сочетании с тем, как ты формулируешь задание после неё. Загрузить бриф и написать «сделай дизайн» это по-прежнему плохой промпт, даже если модель видит документ целиком. Правильная механика выглядит иначе: сначала ты просишь Claude извлечь и структурировать информацию, буквально задать модели роль аналитика, который читает документ и выделяет ключевые элементы, бизнес-цели, ограничения, аудиторию и открытые вопросы. Только после того как эта структура появилась и ты её верифицировал, ты двигаешься к дизайн-задаче. Этот двухшаговый подход предотвращает ситуацию, когда модель делает выводы на основе деталей, которые в реальности второстепенны.

Отдельная история с многостраничными документами: корпоративными брендбуками, исследовательскими отчётами, длинными техническими спецификациями. Здесь работает принцип избирательного фокуса. Загружая документ на сто страниц, имеет смысл явно указать, какие разделы релевантны текущей задаче, иначе модель будет распределять внимание равномерно, а равномерное внимание к ста страницам это почти гарантированно размытый вывод. Опытный дизайнер знает, что из брендбука ему нужны три страницы про голос бренда и раздел про цветовую систему, а не глава про историю компании, и это знание стоит буквально перевести в инструкцию для модели. Ещё один полезный приём при работе с противоречивыми документами, например с брифом, в котором маркетинг хочет одно, а продукт другое, это попросить Claude извлечь информацию и явно обозначить точки противоречий. Модель справляется с этим хорошо и выдаёт на поверхность конфликты, которые в противном случае ты бы обнаружил только на этапе согласования макетов, потеряв несколько итераций.

Figma-в-AI: мосты, плагины и прямые интеграции 2026 года

Figma и AI долгое время существовали в параллельных вселенных, между которыми нужно было совершать ручные переходы: экспортировать кадры, делать скриншоты, описывать компоненты текстом. Это был один из тех рабочих ритуалов, которые кажутся нормальными до тех пор, пока ты не перестаёшь их делать. К 2026 году ситуация изменилась радикально, хотя и не так линейно, как предсказывали аналитики год назад.

Наиболее зрелым мостом между Figma и Claude на сегодняшний день остаётся экспорт через плагин с передачей структурированного JSON-представления файла. Figma давно открыла свой REST API, и экосистема плагинов вокруг него выросла до состояния, когда ты можешь экспортировать не просто визуальное изображение кадра, но полную семантическую структуру: имена слоёв, токены, привязки компонентов, логику автолейаута. Когда Claude получает этот структурированный вывод вместе со скриншотом соответствующего кадра, контекст становится принципиально богаче, чем при работе только с изображением. Модель видит компонент Button/Primary со значением токена color/action/default и состоянием hover, определённым в системе. Это разница между тем, чтобы смотреть на здание снаружи, и тем, чтобы видеть его чертёж.

Прямые интеграции 2026 года, то есть нативная синхронизация между Figma и Claude без плагинного посредника, пока находятся в состоянии активного, но незаконченного формирования. Anthropic и Figma анонсировали партнёрство в области интеграции MCP-коннекторов, но практический путь пока требует настройки и не является историей «поставил и работает из коробки». Это важно признать честно, потому что маркетинговые материалы создают завышенные ожидания. Реальный рабочий процесс для большинства команд в 2026 году выглядит как комбинация: плагин для структурированного экспорта плюс скриншот плюс короткое текстовое описание контекста. Это всё ещё требует нескольких ручных шагов, но их количество сократилось с «много» до «три», что на практике означает принципиально другое ощущение от процесса. Самое ценное, что даёт этот подход: ты можешь передавать Claude живой рабочий файл со всем его реальным состоянием, включая незаконченные части и пометки для себя, а не идеализированный артефакт, который ты специально подготовил для демонстрации. Модель умеет работать с таким материалом и выдавать полезный анализ, а не требовать предварительной «уборки».

Веб-скрапинг как исследовательский инструмент: анализ конкурентов за минуты

Конкурентный анализ в традиционном формате это один из самых трудоёмких и при этом наименее любимых этапов дизайн-процесса. Провести несколько часов, переключаясь между вкладками, делая скриншоты, складывая их в папку и пытаясь потом из этой папки выработать какой-то связный вывод, это занятие, которое одновременно важно стратегически и изматывает тактически. Веб-скрапинг в связке с мультимодальным анализом меняет не саму необходимость исследования, но его механику и скорость настолько радикально, что это меняет и то, сколько исследования ты можешь себе позволить.

Практически это работает в двух режимах. Первый, анализ через скриншоты с Claude: ты систематически снимаешь экраны ключевых состояний конкурентских продуктов, онбординг, главный экран, ключевые пользовательские сценарии, и передаёшь их Claude с конкретным исследовательским вопросом. Вопрос должен быть аналитически заострённым: «выяви паттерны навигационной архитектуры», или «проанализируй, как каждый из этих продуктов решает задачу первичного обучения пользователя», или «найди расхождения в том, как эти интерфейсы работают с пустым состоянием». Конкретный вопрос даёт конкретный аналитический вывод, а не описательный перечень наблюдений. Второй режим, работа с Claude через браузерные MCP-коннекторы, которые позволяют модели напрямую просматривать публичные веб-страницы в режиме реального времени. Это мощнее, потому что снимает необходимость ручного сбора скриншотов, но требует настройки и работает надёжно только с публично доступными страницами без жёстких антибот-защит.

Конец ознакомительного фрагмента.