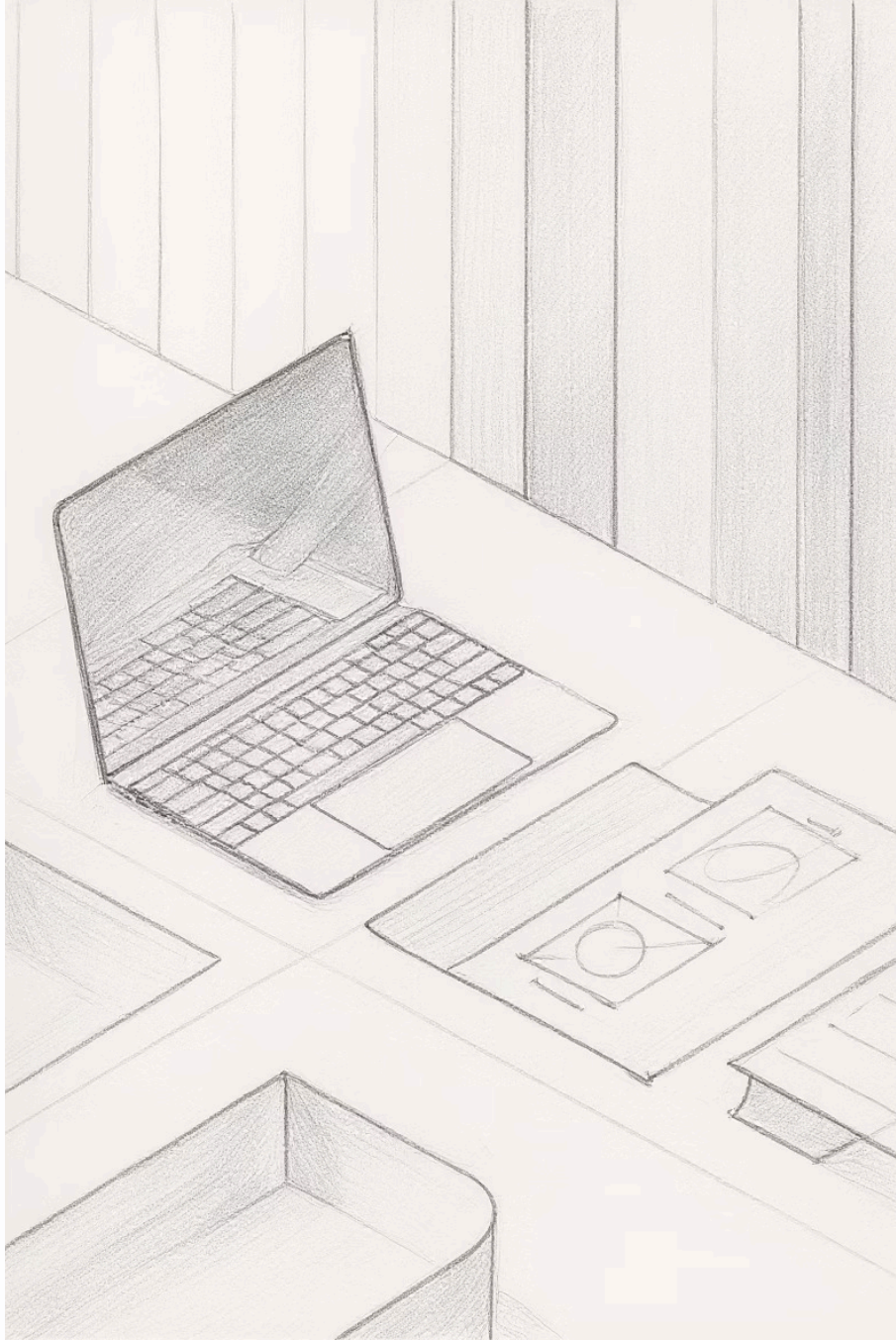
Module 1: Scoping Reality

- Find what needs to be true for your business to work
- Cut features ruthlessly
- Test risky assumptions first
- Scope an MVP that validates your business model.



MVP vs Prototype vs Market-Ready

Some founders confuse these. Here's what they actually mean.

Prototype

$5K, 2-3 weeks

- Tests if concept makes sense
- NOT sellable to customers
- Clickable mockups or manual processes

Learn: Do people understand? Would they pay?

MVP

$20K-50K, 2-4 months

- Actually works, functional
- Real customers paying real money
- Basic features only

Learn: Will people pay? Do they use it repeatedly?

Market-Ready

$60K-150K, 4-8 months

- Competes with existing solutions
- Polished user experience
- Ready for paid marketing at scale

Need: After MVP proves product-market fit

⚠️ **Critical mistake:** Most founders think they need Market-Ready for launch. They most often need MVP.

Prototype Deep Dive

What It Is

- Tests if concept makes sense
- Held together with duct tape
- Used for user research

Examples

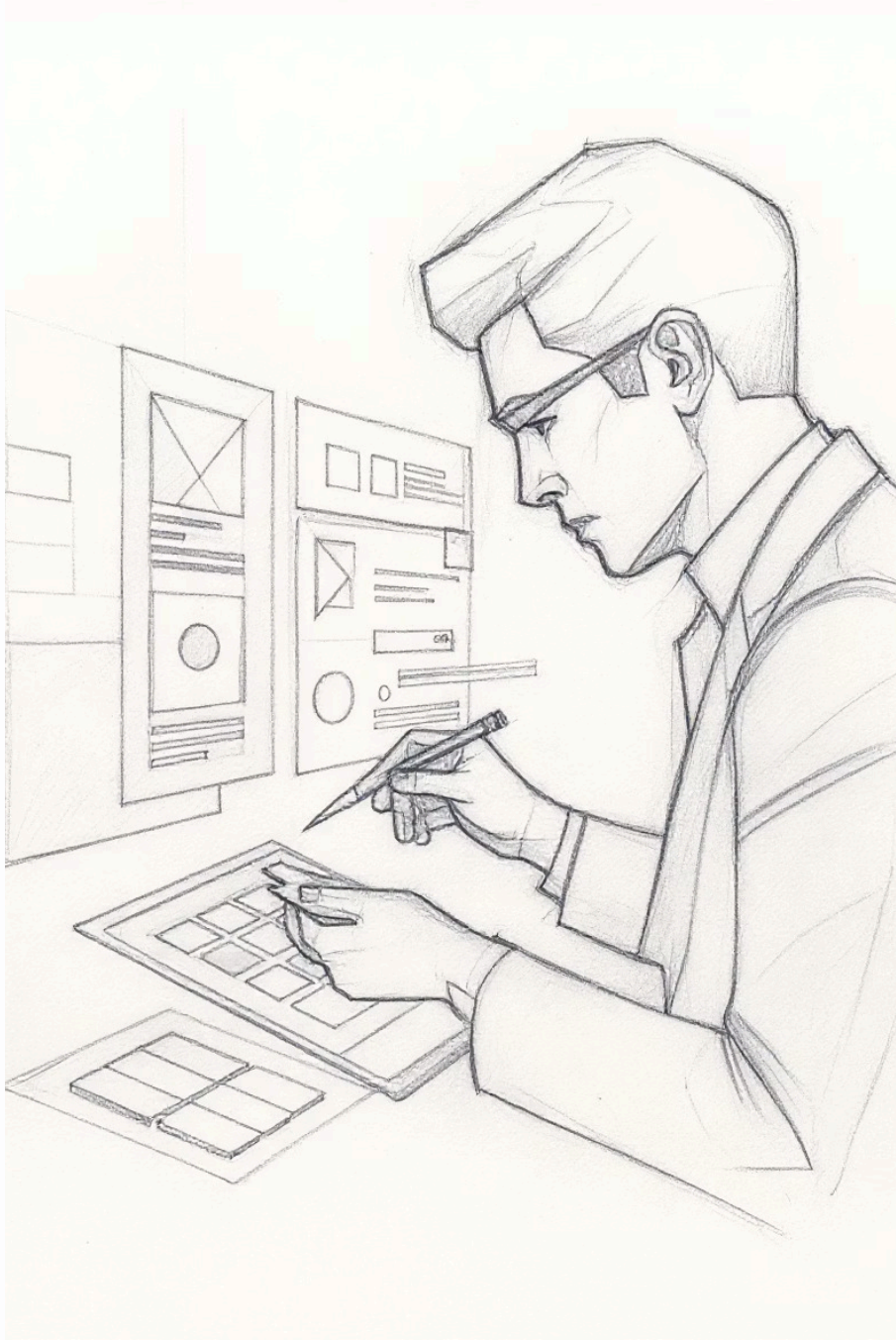
- Clickable Figma mockups
- Wizard of Oz manual fulfillment
- Landing page + email signup
- Manual concierge service

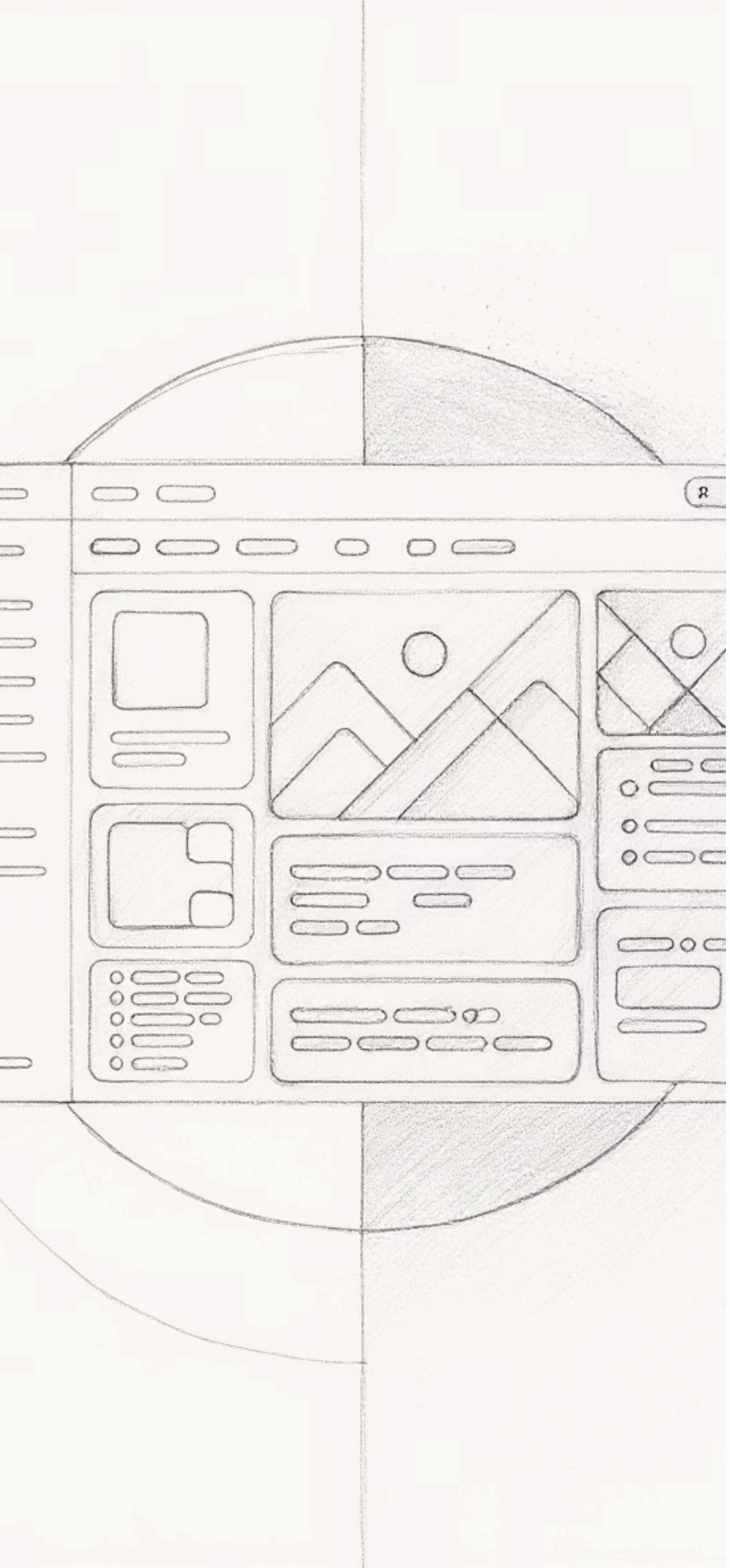
What You Can Learn

- Do people understand?
- Would they pay?
- What features matter?
- Is UI/UX intuitive?

What You Can't Learn

- Technical feasibility at scale
- Real user behavior over time
- Actual conversion rates





MVP

($20K-50K, 2-4 months)

What It Is

An MVP is a functional product.
It delivers core value to early users.

- It actually works.
- For customers paying money.
- Focuses only on core features.

Real-World Examples

- **Airbnb v1:** Founders' own apartment for rent, manual processes for bookings and payments, personal photos.
- **Instagram v1:** Included photo upload, filters, and a feed; no video, DMs, stories, or comments.
- **Dropbox v1:** Basic file sync for a single folder, only Windows.

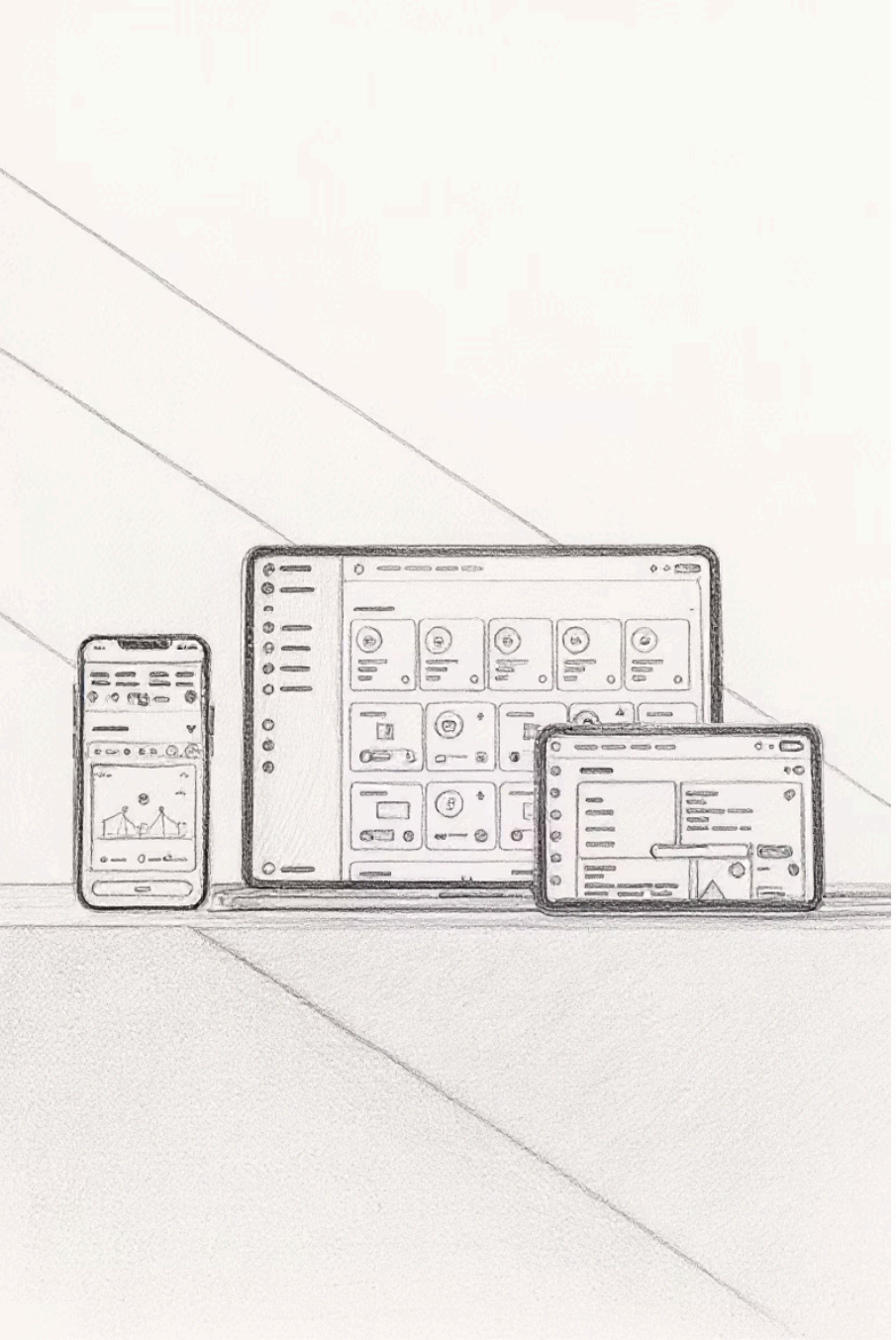
Key Learnings

- Verify if people will pay.
- Understand if users return.
- Identify the conversion funnel and its bottlenecks.
- Determine specific features driving user retention.

What You Sacrifice

- Beautiful and polished design.
- Handling of all edge cases or errors (only critical).
- Scalability beyond a few thousands users.
- Features that competitors may offer.

⚠️ **Critical distinction:** An MVP involves real customers paying real money, whereas a prototype typically does not.



Market-Ready Products

What It Is

- Competes directly with existing solutions
- Polished user experience
- Handles edge cases gracefully
- Ready for paid marketing at scale

When You Need This

- After MVP proves product-market fit
- When you have revenue to invest
- Competing in crowded market
- Targeting skeptical buyers

Decision Tree: What Should You Build?

1

Do you have paying customers?

Yes → Skip prototype, build MVP or Market-Ready

No → Keep reading

2

Is your concept brand new?

Yes → Start with Prototype

No → Skip to MVP

3

Do you have $20K+ budget?

Yes → Build MVP

No → Build Prototype first

4

Are you in a crowded market?

Yes + $60K+ → Consider Market-Ready

Yes + $20-50K → MVP with excellent UX

No → MVP is fine

Path 1: Fast Validation MVP

$20K-40K, 2-3 months

Choose This If:

- Testing product-market fit
- Need to validate demand quickly
- Want to run ads/marketing tests
- Have limited budget
- First-time founder or unproven market

Strategy

Reuse everything possible

Examples

- Mobile app testing conversion through ads
- SaaS platform to validate with existing network
- Marketplace testing supply/demand dynamics

Trade-off

Less defensible, easier to copy

Why It Works

Speed matters more than perfection when you're learning

Path 2: Platform/Differentiation MVP

$50K-80K+, 4-6+ months

Choose This If:

Building unique technology that IS your moat (defensible competitive advantage)

Have resources and proven market

Strategy

Build custom where it matters

Examples

- Recommendation algorithms
- Proprietary matching systems
- Unique data processing
- Novel UX patterns

Trade-off

Higher cost, longer timeline

Path Selection Scorecard

Score each question 1-5:

Question	Score (1-5)
How proven is your market?	1=brand new, 5=established
How much budget do you have?	1=$20K, 5=$100K+
How important is speed?	1=not urgent, 5=critical
How easy to copy is your idea?	1=very easy, 5=very hard
How technical is your differentiation?	1=not technical, 5=core tech

01

5-15 Points

Path 1 (Fast Validation)

02

16-20 Points

Either path works, depends on goals

03

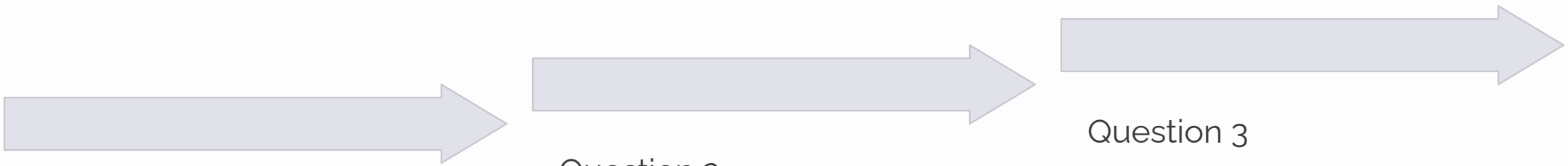
21-25 Points

Path 2 (Differentiation) might be justified

> **Reality check:** 70% of founders should choose Path 1.

Feature Cutting: The 3-Question Filter

Often founders want to build too many features in MVP. Here's how to cut ruthlessly.



Question 1

Can users accomplish their core goal without it?

Yes → Defer to post-MVP

No → Continue to Q2

Question 2

Can we fake this for first 10-50 customers?

Yes → Use Wizard of Oz approach, defer building

No → Continue to Q3

Question 3

Does this test a risky assumption?

Yes → Build it

No → Defer to post-MVP

Wizard of Oz approach: Users interact with what seems like a fully functional system, but parts are actually operated manually behind the scenes by humans.

Feature Decision Table

List every feature you want and evaluate:

- [ ] P0 = Must have for MVP

  Build first

- [ ] P1 = Should have if budget allows

  Build second

- [ ] P2 = Nice to have

  v1.1 after launch

- [ ] P3 = Not needed yet

  v2+

> **Goal:** Only build P0 features in your MVP.

Feature Decision Table

List every feature you want and evaluate:

Feature	Core Goal?	Can Fake?	Tests Risky Assumption?	Decision	Priority
User authentication	Yes	No	No	Build (basic)	P0
Onboarding	No	Yes	Yes (will people want to register?)	Build (basic)	P0
Password reset	No	Yes (manually)	No	Defer	P2
Social login (Google/FB)	No	N/A	No	Defer	P3
Payment processing	Yes	No	Yes (will people pay?)	Build (Stripe)	P0
Saved payment methods	No	Yes (re-enter)	No	Defer	P2
Subscription management	Yes (for SaaS)	Partially	Yes	Build (basic)	P0
Invoice generation	No	Yes (manual)	No	Defer	P3
Email notifications	No	Yes (manual)	No	Defer	P2
In-app messaging	No	Yes (email)	No	Defer	P3
User profile	Maybe	No	No	Bulid (minimal)	P1
Profile photos	No	Yes (initials)	No	Defer	P3
Settings page	No	Yes (contact us)	No	Defer	P2
Search	Depends	Yes	Maybe	Evaluate	P1-P2

Your Feature List Exercise

01

List Features

List every feature you think you need

02

Fill Decision Table

Complete the evaluation for each feature

03

Count P0 Features

- 3-5 features → Good scope
- 6-8 features → Probably too many, review
- 9+ features → Definitely too many, cut more

04

Final Check

For each P0: "If I removed this, would the MVP still test my riskiest assumption?"

Yes → Move to P1 | **No** → Keep as P0

Reality check: Instagram launched with 3 features: upload photo, apply filter, share to feed. That's it.

Feature	Value (1-10)	Complexity (1-10)	ROI	Tests Risk?	Can Fake?	Decision
Recommendation engine	10	9	1.1	Yes	Partially	Build (V1)
Payment processing	10	3 (Stripe)	3.3	Yes	No	Build
User authentication	7	4	1.8	No	No	Build (bask
Email notifications	6	3	2.0	No	Yes	Defer
Social sharing	4	5	0.8	No	Yes	Defer
Admin dashboard	5	7	0.7	No	Yes	Defer
Real-time chat	6	8	0.75	No	Yes (email)	Defer
Push notifications	5	4	1.25	No	Yes	Defer
Analytics	4	6	0.67	No	Yes	Defer
Advanced search	5	7	0.71	No	Yes	Defer

The 80/20 Analysis

Instructions

1. Rate value to users (1-10)
2. Rate build complexity (1-10, ask developer)
3. Calculate ROI (value / complexity)
4. Sort by ROI descending

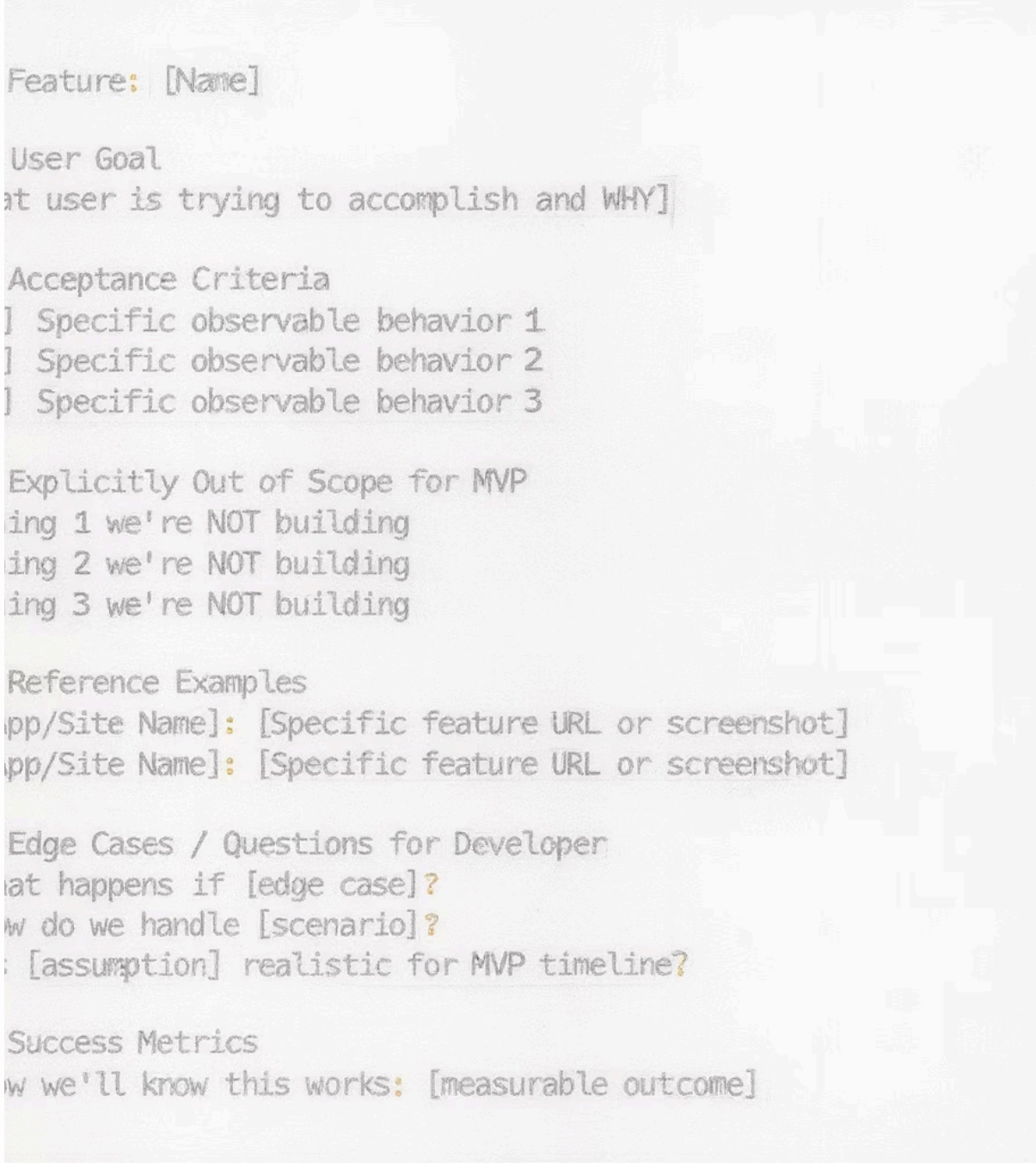
Decision Rules

- ROI > 2.0 + Tests Risk = **Build**
- ROI > 2.0 + Can't Fake = **Consider building**
- ROI < 1.5 = **Defer** unless tests critical assumption
- Can Fake = **Always defer** unless tests risk

> **Insight:** The features you're most excited about often have the worst ROI.

User Story Templates: Developer-Ready Format

Developers can't work from "As a user, I want to see my order history." They need specifics.



Example: Bad vs Good User Story

❌ BAD

(what founders write)

> "As a user, I want to see my order history."

✅ GOOD

(what developers can work from)

Feature: Order History View

User Goal: Check status of recent orders without calling support, see past orders to re-order items.

Acceptance Criteria

- View list of all orders (most recent first)
- Each order displays: date, items (first 3 + count), total, status
- Click any order to see full details
- Status shows: Processing, Shipped, Delivered, Cancelled
- Status updates when user refreshes page

Out of Scope for MVP

- Filtering by date range
- Searching orders
- Exporting to PDF
- Cancelling orders from history page
- Sorting options
- Tracking numbers (show on detail page only)

Reference Examples

- Amazon order history (desktop)
- DoorDash order tracking (mobile app)
- Grubhub order history: Simple list view

Reuse vs Build Custom: The Decision Framework

⚠️ Never Build These Yourself (Use Existing Services)

Prioritize your development efforts on what makes your product unique.

Payment Processing	Stripe, PayPal, Square	PCI compliance, security, trust, global reach	$15K-30K + liability
Authentication (Basic)	Firebase Auth, Auth0, AWS Cognito	Security, password reset, 2FA, scalability	$5K-10K
Email Sending	SendGrid, Mailgun, AWS SES	High deliverability, spam management, analytics	$8K-15K + poor delivery
SMS Notifications	Twilio, AWS SNS	Carrier relationships, delivery reports, global coverage	Impossible without contracts
File Storage	AWS S3, Cloudinary, UploadCare	CDN, backups, scaling, security, metadata	$5K-10K
Push Notifications	Firebase Cloud Messaging, OneSignal	Device management, delivery, cross-platform support	$10K-20K
Analytics (Basic)	Google Analytics, Mixpanel, Amplitude	Pre-built dashboards, reports, robust tracking	$15K-30K
Maps/Geolocation	Google Maps, Mapbox	Accurate data, geocoding, routing, global coverage	Millions + impossible

When to Build Custom

✅ Build custom if:

- It's your core differentiation (e.g., a recommendation engine, matching algorithm).
- It tests a risky assumption (e.g., a novel UX pattern, unique workflow).
- No existing service perfectly does what you need (this is rare).
- The integration cost of a third-party service significantly outweighs the cost of building it yourself (this is very rare).

❌ Don't build custom because:

- "We might need custom features later" (you can migrate later).
- "I don't want to pay 3% fees" (your development time costs more).
- "We want full control" (you'll likely never fully utilize that control).
- "It's not that hard to build" (it's harder than you think).

Build vs Buy Decision Table

Apply the framework to your features. Here's an example of how you might evaluate different components for your MVP.

Feature	Buy Option	Buy Cost	Build Cost	Tests Risk?	Decision	Rationale
Payments	Stripe	3%	$20K	No	Buy	Security, compliance, no-brainer
Auth	Firebase	Free	$5K	No	Buy	Standard solution works
Email	SendGrid	$10/mo	$10K	No	Buy	Deliverability matters
Admin UI	Retool	$50/mo	$15K	No	Buy	Internal tool, doesn't need beauty
Recommendations	None good	N/A	$30K	Yes	Build	Core differentiation
Matching logic	None	N/A	$25K	Yes	Build	Unique algorithm is the moat
Chat	Stream/SendBird	$100/mo	$20K	No	Buy	Commodity feature
Video calls	Twilio/Agora	$0.004/min	$40K	No	Buy	Complex infrastructure

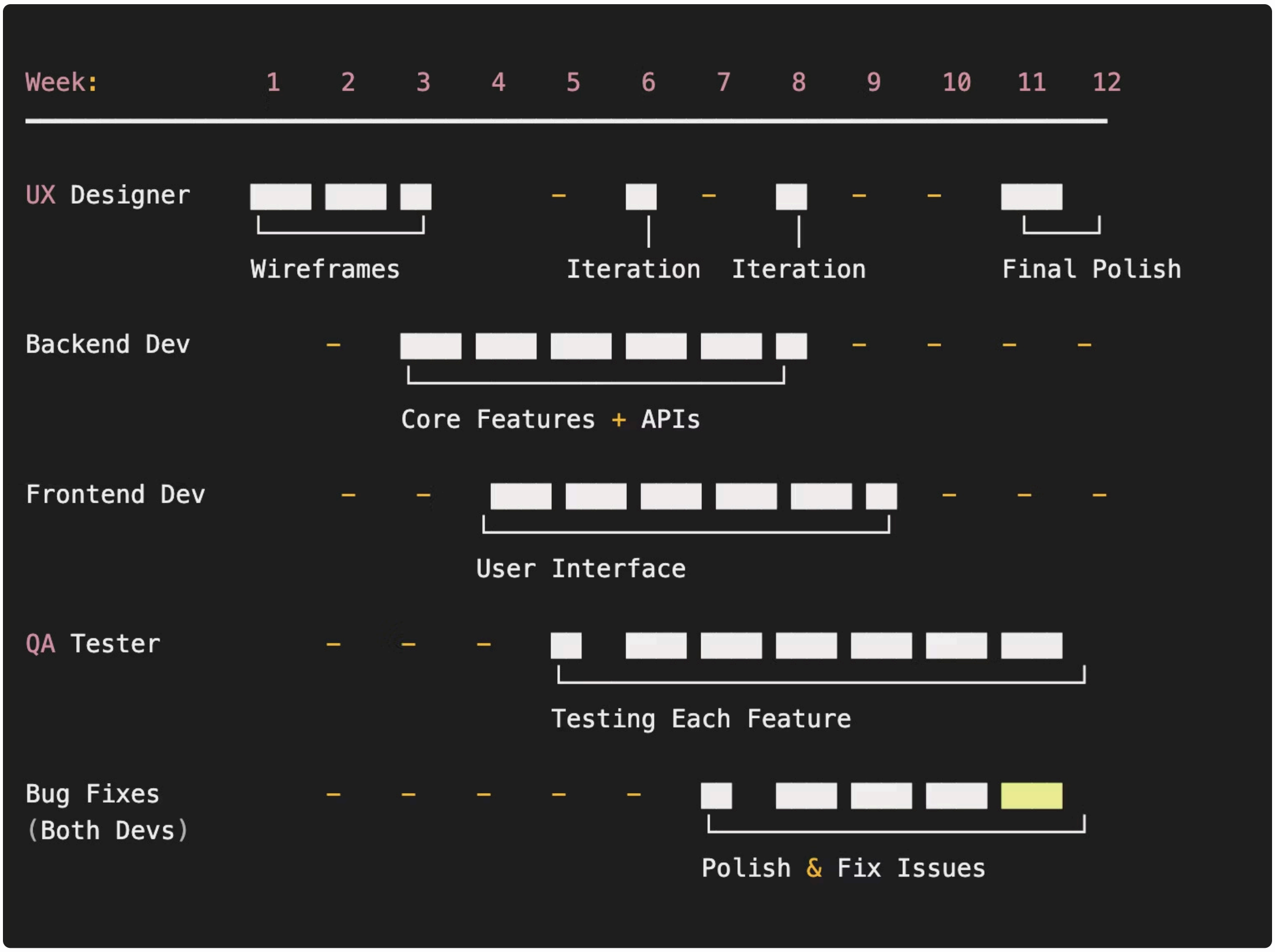
This table helps you systematically decide where to invest your development resources, prioritizing unique value while leveraging existing solutions for commodity features.

What a Real Delivery Plan Looks Like

This example shows a **12-week Task Management MVP** with realistic parallel work. Team: 2 Developers, UX Designer, QA Tester

Project: Simple Team Task Board (Trello Competitor)

This high-level overview sets clear expectations for scope, resources, and timelines before diving into detailed feature breakdowns and development sprints.



Week-by-Week Breakdown

A detailed plan outlining key activities, team responsibilities, and deliverables for each two-week sprint, leading to a fully functional MVP.

01

Weeks 1-2: Foundation

UX Designer: Wireframes, user flows, design system
Backend Dev: Project setup (DB, API, auth)
Frontend Dev: Not started
QA: Not started

Deliverable: Approved designs + working authentication

02

Weeks 3-4: Feature 1 - Task Creation & Lists

Backend Dev: APIs for tasks/lists
Frontend Dev: Task forms, list views, basic editing
UX Designer: Design iterations
QA (starts Week 4): Tests task creation, bug finding

Deliverable: Users can create and organize tasks

03

Weeks 5-6: Feature 2 - Team Collaboration

Backend Dev: Workspace, invitations, permissions
Frontend Dev: Team dashboard, invitation UI
UX Designer: Improves invitation flow
QA (parallel): Tests workspace, retests Feature 1 fixes

Deliverable: Multiple users can work together

04

Weeks 7-8: Feature 3 - Real-time Updates

Backend Dev: WebSocket, live data sync
Frontend Dev: Real-time UI updates
UX Designer: Polishes loading states
QA (parallel): Tests real-time sync, regression testing
Bug Fixes: Developers address critical issues

Deliverable: Changes appear instantly for all team members

05

Weeks 9-10: Features 4 & 5 - Comments & Due Dates

Backend Dev: Comment API, due date tracking
Frontend Dev: Comment threads, date picker
UX Designer: Final consistency pass
QA (heavy load): New feature tests, full regression
Bug Fixes: Developers fix QA findings

Deliverable: Complete task communication system

06

Weeks 11-12: Feature 6 - Payments & Launch Prep

Backend Dev: Stripe integration, subscription logic
Frontend Dev: Pricing page, checkout, billing
UX Designer: Final polish, onboarding, email templates
QA (final push): Payment testing, security, performance
All Team: Bug bash, demo video, documentation

Deliverable: Production-ready MVP with payment

Why This Schedule Works

This structured approach ensures efficiency, quality, and a predictable path to launch.

Work Happens in Parallel

This prevents idle time and maximizes team productivity.

Buffer Time Included

Dedicated periods for bug fixes and final polish are built into the schedule.

Issues are caught and resolved early.

Clear Handoffs

Dependencies are managed with precision.

Smooth transition between stages prevents bottlenecks and rework.

Weekly Review Checklist

Regular, targeted reviews are crucial to keeping your MVP development on track and validating assumptions early. Use this checklist to guide your weekly check-ins and identify potential red flags.

Week	What to Check	Red Flag If...
2	Review wireframes - does this make sense?	You're confused by the user flow
4	Test task creation yourself	Core interaction feels clunky
6	Invite a friend, test together	Collaboration is confusing
8	Test on 2 devices simultaneously	Updates don't appear instantly
10	Use it for your own tasks for 2 days	You wouldn't use this yourself
12	Would you pay $10/month for this?	Honest answer is "no"

By actively engaging with the product at each stage, you can catch critical issues early, ensuring the final MVP aligns with user needs and business goals.

What to Cut If Timeline Slips

Even with the best planning, project timelines can shift.

If 1 Week Behind (finish Week 13)

- Real-time updates (5-second auto-refresh)
- Onboarding tooltips

Keep: All 5 core features working

If 2 Weeks Behind (finish Week 14)

- Comments (collect via email)
- Due date reminders

Keep: Basic task management + teams + payments

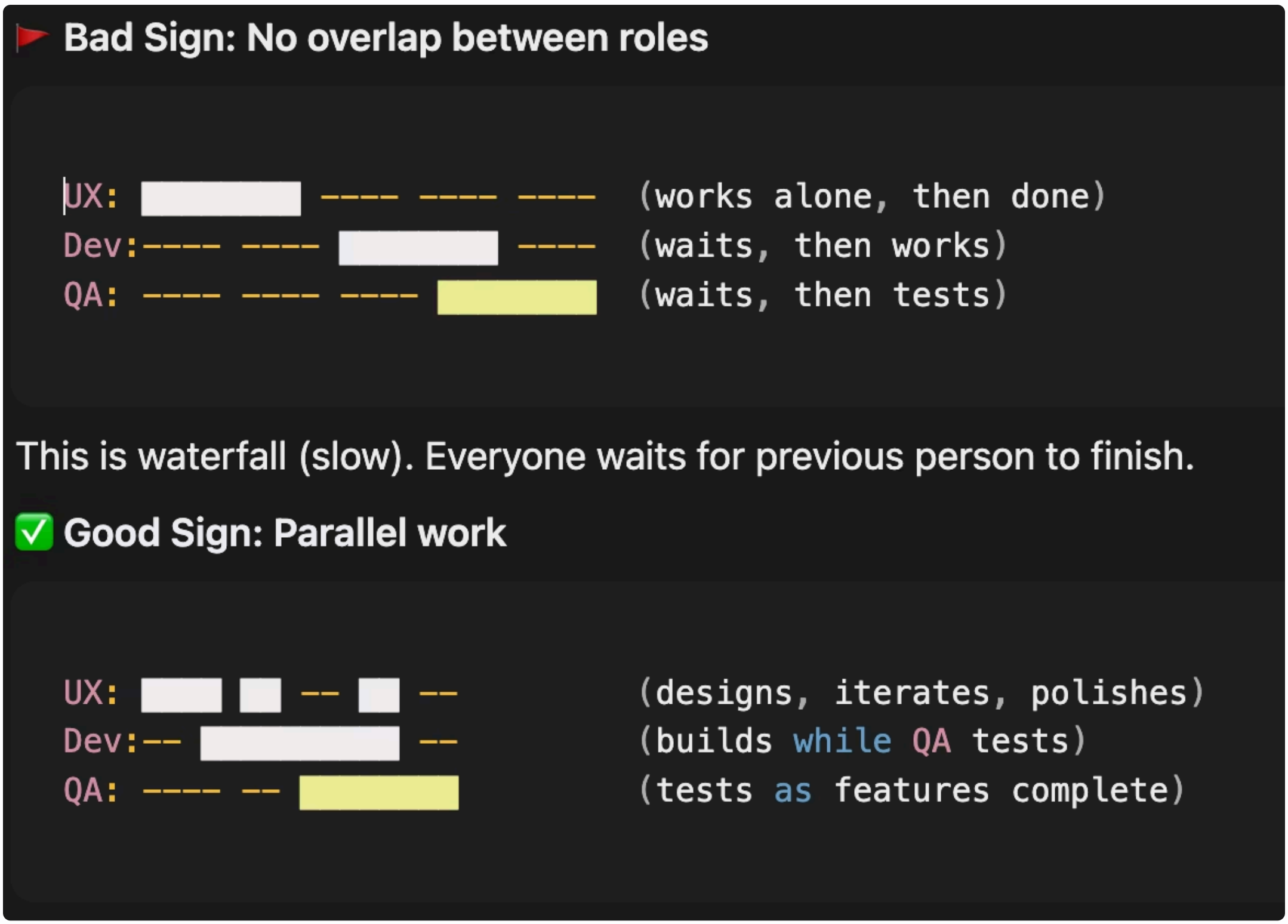
If 3+ Weeks Behind

Stop. Something is fundamentally wrong.

Meet with team to understand why (bad estimates? scope creep? technical issues?).

Consider: Reduce scope to 3 features or find a new team.

Common Delivery Plan Red Flags



Watch out for these warning signs. Recognizing them early can save significant time and resources.

- 🚩 "Testing phase" at the end

  If QA only starts in Week 10-12, you'll find major problems too late to fix.

- ✅ QA starts mid-project

  QA starts testing Feature 1 while developers build Feature 2.

- 🚩 No buffer time

  A plan that shows 100% feature work with 0% time for bug fixing is unrealistic .

- ✅ Buffer built in

  Last 3-4 weeks explicitly include time for bug fixes, polish.

Questions to Ask Your Developer/Agency

Equip yourself with questions to identify potential red flags in their development approach.

Show me Week 5 - What is everyone doing?

Good: Everyone has clear tasks, and work streams overlap efficiently.

Bad: "Well, it depends..." (indicates a lack of detailed planning).

When does QA start testing?

Good: "Week 4, after the first feature is done."

Bad: "At the end" or "We don't include QA" (risky and costly).

What happens if you find a major bug in Week 8?

Good: "We have 3-4 weeks of buffer time specifically for fixes."

Bad: "We'll just push the timeline" (a recipe for scope creep).

Who reviews the work each week?

Good: "You see demos every Friday for direct feedback."

Bad: "We'll show you when it's done" (a significant red flag for transparency).

What's included in the QA testing?

Good: "Functional, cross-browser, mobile responsive, security basics."

Bad: "We'll make sure it works" (vague and lacks specifics).

This Is How Professionals Plan

Realistic Timelines

Built-in buffer time for unforeseen challenges and thorough bug fixing.

Parallel Workflows

Concurrent development, design, and QA maximize efficiency.

Early & Continuous Testing

QA starts early, catching issues before they escalate.

Regular Progress Demos

Consistent demos provide timely feedback and alignment.

Transparent Costing

Clear financial commitments offer predictability and avoid surprises.

Avoid those who "figure it out as we go."

Your next steps

Grab template of

- Scope document
- Estimation (add features you want to build)

Fill it with your project details.
Get hands dirty.

Developers will help to make these documents solid. You need some practice first.