

// phase 3 deep dive

Problem Selection & Expert Depth



How to pick the one problem your target companies share, then out-research everyone on it, so your project reads like it was built by an insider.

baswe.ai // become a software engineer

This is the in-depth guide to phase three of the Signal Project. Work through it once you have your signal log and a set of recurring problems from phase two. By the end you will have chosen one problem to build for, and become the person who understands it best.

Why this is the phase that decides everything

Phase two handed you a list of candidate problems. This phase is where you do two things with them: choose the one your project will solve, and then go deep enough to understand it better than almost anyone else applying to these companies. That choice, plus that depth, is your project. A well-chosen problem you deeply understand makes the build straightforward and the launch compelling. A vague problem you barely understand makes everything after it weak, no matter how clean your code is.

This is also the phase that most directly proves the one thing companies actually want. They are not looking for someone who can code. They are looking for someone who understands their problems and can deliver value. This is where you prove you understand a real problem from the inside.

Two moves, in order: first, pick the right problem from your research. Second, out-research everyone on it. Selection makes your project relevant. Expert depth makes it credible.

Part one: choosing the problem

From your phase two signal log, you should have several candidate problems and trends that showed up across your target companies. Your job now is to narrow those to a strong shortlist, then commit to one.

▪ Start from your signal log

Pull the recurring signals out of your research and group similar ones together. You are looking for three to five real problems or trends, with a bias toward the ones that appeared at multiple target companies. A problem several of your companies share is worth more than one that only shows up at a single company, because solving it makes your project relevant to a whole segment at once.

▪ What makes a good problem to build for

Real and specific. An actual pain a company has, not a vague theme. "Companies are using AI" is a theme. "Support teams cannot quickly find answers buried in their internal docs" is a problem you can build for.

Shared across multiple targets. If several of your companies have it, one project speaks to all of them, and your outreach and launch reach a whole group instead of one company.

Buildable in one to two months. You, or a small team, can ship a meaningful version in that window. If it clearly needs a year, it is too big. Scope it down until it fits.

Reveals domain understanding. Solving it should show that you actually understand the industry, not just that you can write code. That understanding is the signal.

Genuinely interesting to you. You are about to spend weeks on this and talk about it in public. Pick something you can actually care about, or it will show.

- **Pick the un-obvious one**

This is the contrarian move that matters most, and it comes straight from the playbook. Do not automatically grab the flashiest, most popular trend. Very often the second, third, or fourth problem on your list is the better choice. It is usually less saturated, more achievable in your timeframe, and it proves a deeper understanding of the industry, because knowing about it means you actually understand that world, not just the headlines.

The logistics example: a small project built around a bill of lading, an unglamorous document that is still handled on paper at many companies, pulled serious attention from logistics startups and large companies, got more recruiter eyes, and passed more applicant screens, all because that one specific, obscure term signaled real domain understanding. The obscure but real problem beat the flashy one.

- **Narrow to one with a simple scorecard**

To choose between your three to five candidates, score each one against the criteria above: real and specific, shared across targets, buildable in one to two months, reveals domain depth, interesting to you, and un-obvious enough to stand out. The candidate that scores well across the board, especially on buildability and domain depth, is your pick. Do not agonize over it. A good problem you actually execute beats a perfect problem you never start. Note your runner-up in case you need to pivot later.

- **Sanity-check before you commit**

Before you lock it in, run four quick gut-checks. Can you describe the problem in a single sentence? Can you name at least three target companies that have it? Can you picture the small version you would build to address it? Can you realistically ship that in one to two months? If the answer to all four is yes, commit. If not, tighten the scope or pick another candidate.

Part two: becoming the expert

You have your problem. Now spend one to two weeks becoming a mini expert on it. This is the research camp, and it is the difference between a project that looks like a class assignment and one that looks like it came from someone who lives in the industry. The goal is simple: understand this problem better than the average engineer

applying to these companies. That depth is what makes your build sharp, your launch story compelling, and your interviews easy.

Think of it the way a founder would. Before building, founders get deep on the problem they are solving. You are building a small MVP of a startup, so you do the same. Companies do not want proof you can code. They want proof you can deliver value, and value comes from understanding the problem.

▪ **Run a focused research sprint**

Over one to two weeks, work through these questions about your chosen problem and capture the answers as you go:

- ▶ Why does this problem exist? What actually causes it?
- ▶ Why has no one solved it well yet? Is it cost, complexity, regulation, incentives, or that nobody has cared enough?
- ▶ Why does it matter to the company, now and in the future?
- ▶ What solutions already exist? Who are the players and tools?
- ▶ Where do those solutions fall short, and where do they work well?
- ▶ How expensive or cheap are the current approaches?
- ▶ What would an ideal solution actually look like?
- ▶ How does solving this fit the company's bigger goals and the industry's direction?

Answering these turns a coding exercise into something that looks and feels like a real product, and it hands you the substance you will use in your launch article and your interviews.

▪ **Where to do this research**

Go deeper on this one problem than you did on any single company in phase two. Use:

- ▶ **Competitor products:** sign up, actually use them, and note exactly where they fall short.
- ▶ **Their docs and pricing pages:** these reveal how the problem is framed and what it costs to solve today.
- ▶ **Industry reports and analyst pieces:** for the size, shape, and direction of the problem.
- ▶ **Communities and forums:** Reddit, industry Slacks and Discords, and LinkedIn groups, where the people with the problem talk about it honestly.
- ▶ **Company engineering blogs and job posts:** they often describe the exact problem in the company's own words.
- ▶ **Podcasts and talks from people in the space:** for how insiders think and the vocabulary they use.

The aim is to become fluent in how insiders talk about this problem. When you use their language, everything you build and say lands as credible.

▪ Talk to real people

This is the highest-leverage move in the whole phase, and it is where your silent network from phase two finally pays off. Message the people you connected with, but frame it correctly. You are not asking for a job, and you are not asking for a referral. You are building this product with or without them, and you would value their expertise. Reference something real, like a talk their CEO gave or the work their team is doing, and ask for their perspective on the problem itself.

Here is the framing to use:

OUTREACH MESSAGE FRAMING (THIS IS NOT A PITCH)

Open. Reference how you are connected, or something specific about them or their company.

Context. "I'm building [X] to tackle [the problem]. I've been digging into it, and I saw [their CEO's talk / your team's work / what is happening in the space]."

Ask. "Would you be open to 15 minutes to share how you and your team think about [the problem]? I'm building this with or without help, I just want to learn from people who actually work in it."

Rules. Keep it short, be genuine, reference something real, and do not ask for a job or a referral.

This works because people like helping, especially when the message is genuine and not a sales pitch. You get real advice from people who live the problem, which makes your product far better, and you plant a seed. Months later, when you apply, they remember you as the person who built that thing and talked to them about it, which makes them much more likely to refer you or push you to their manager.

▪ Capture it all in a research brief

As you research and talk to people, write it down in a short brief: the problem in your own words, why it exists, the current solutions and where they fall short, what your solution will do differently, the key facts and numbers, and the best insights and quotes from your conversations. This brief becomes the backbone of your MVP plan in the next phase and your launch story later, and it is your ammunition in interviews. Do not keep it all in your head.

How the two halves connect

You picked a problem your target companies share, then went deep enough to understand it better than most applicants. The selection made your project relevant to a whole segment. The expert depth made it credible. Now you can build something that actually matters, and talk about it like an insider, which is exactly what the next phases turn into a shipped product and interviews.

Common mistakes to avoid

- ▶ **Chasing the flashiest trend.** The most popular problem is usually the most saturated, the hardest to build, and the least differentiating. The un-obvious one wins.
- ▶ **Picking a problem too big to build.** If it needs a year, it is the wrong scope. Shrink it until you can ship in one to two months.
- ▶ **Choosing a vague theme instead of a specific problem.** You cannot build for "AI in healthcare." You can build for one concrete pain inside it.
- ▶ **Skipping the expert-depth work.** Building on a shallow understanding shows up in your project and your interviews. Do the research camp.
- ▶ **Researching entirely alone.** If you never talk to real people, you miss the best insights and lose the referral seeds. Use your network.
- ▶ **Framing outreach as "I want a job."** That kills the willingness to help. Ask for expertise, not employment.
- ▶ **Never committing.** Endlessly researching problems instead of choosing one is just a way to avoid building. Pick one and go deep.
- ▶ **Writing nothing down.** The substance you gather now is what powers the build and the launch. Capture it in a brief.

What you should have before moving on

By the end of phase three, you should have:

- ▶ One chosen problem you can describe in a single sentence.
- ▶ At least three target companies that clearly have that problem.
- ▶ A research brief covering why the problem exists, the current solutions and their gaps, and what an ideal solution looks like.
- ▶ Real insights from at least a few conversations with people in the space, and warm connections for later.
- ▶ A rough idea of what your MVP needs to do, ready to be sharpened in the next phase.

That problem and that brief are exactly what the next phase needs. Do not move on until you can state your problem in one sentence and explain, from real understanding, why it matters and why it has not been solved well.

→ **With your problem chosen and your research brief in hand, move on to Phase 4: Architecture.**