The AI Execution Guide

From Problem to Product

Ignatious P.

Business Analyst and Product Owner
MBA | CSM | CSPO | 9+ Years Fintech

Contact: ignatious@catalystagile.com

Website: catalystagile.com

"The gap between what teams need and what delivery builds is almost always a clarity problem upstream."

A complete, step-by-step framework guiding you on how to use AI and benefit from it, entirely free of cost. Proven across 54 live experiences.

Contents

1 Why This Guide Exists

Most AI advice is vague. This guide is not.

I built this guide for practitioners who know AI can help but are tired of generic prompts, bloated platforms, and polished output that does not survive real work. The goal of this document is simple: to guide you on how to use AI to achieve BRD-precision specifications and benefit from it, completely free of cost.

The gap between what teams need and what delivery builds is almost always a clarity problem upstream. I turn operating need into shipped product. The method in these pages is the exact workflow I use to turn messy problems into usable requirements, clear prompts, tested outputs, and decisions that move work forward.

While others were adopting generic platforms, I was building AI infrastructure. People do not struggle with AI because the models are weak. They struggle because the instruction is vague, the goal is fuzzy, and no one defines what good looks like before they start.

I wrote this guide to fix that. It accelerates the quality of the instruction you give, helping you bridge the gap in your own delivery lifecycle.

2 The Process: Five Steps from Problem to Product

Not magic. Not automation. A repeatable process for turning ideas into tested solutions in days. I used this process for 9+ years as a Business Analyst and Product Owner—decomposing problems, defining acceptance criteria, testing with real users, and iterating until it works. AI did not change the process. It removed the wait.

1. **Identify the Problem:** Not “we need a feature.” A real pain point. Can you explain it in one sentence? If not, you do not understand it yet.
2. **Define Success:** What does “solved” look like? Not vague aspirations. Specific, testable outcomes. Would you know success if you saw it?
3. **Prototype with AI (1-2 days):** Describe what you want clearly. Test what comes back. Iterate. Average: 10-20 iterations per live experience, 5-20 minutes each.
4. **Test with Real Users (1 day):** Show it to 3-5 colleagues. Do not explain it. Watch. User confusion is a design problem, not a user problem.
5. **Ship or Kill (immediate):** No attachment to ideas. Attachment to outcomes. If it works, share it. If it does not, try a different problem.

3 Problem Identification Framework

The most important step. Get this wrong and everything else is wasted effort.

The One-Sentence Test: Can you explain the problem to a colleague in one sentence? If not, stop. You do not understand it yet.

3.1 Questions to Ask Before Building

Question	Why It Matters	Good vs. Bad
Who has this problem?	Specificity determines relevance.	**Good:** "BAs on teams with 8+ devs." **Bad:** "BAs."
How often does it happen?	Frequency equals value.	**Good:** Daily or weekly pain. **Bad:** Once a quarter.
What do they do today?	Reveals the gap between current state and desired state.	**Good:** Manual spreadsheet workaround. **Bad:** "Nothing."
What does "solved" look like?	Makes success testable.	**Good:** Specific criteria. **Bad:** "Better" or "faster."
Would you use this yourself?	Gut check on real value.	**Good:** Yes, immediately. **Bad:** "It is interesting."

3.2 The Problem Statement Template

"[Role] spends [time/effort] on [task] because [root cause]. A solution that [desired function] would save [measurable outcome]."

Real Examples From My Work:

- **Sprint Planning:** "Sprint planning takes 4 hours because we cannot split oversized user stories consistently."
- **Stakeholder Management:** "A massive volume of stakeholders on a banking merger with no clear communication strategy leading to missed updates and escalations."
- **Communication:** "Diplomatic emails take 2 hours because I have to turn frustration into professionalism."

4 How to Describe What You Want (The Architecture)

You do not need to learn to code, but you do need to think like an architect. The live experiences on catalystagile.com are not static forms; they are dynamic engines. To get AI to build at that level, you need to describe what you want the same way you would write a comprehensive BRD—clearly, with acceptance criteria, variable logic, and strict output parameters.

4.1 The Five-Part Prompt Formula for Dynamic Engines

If you want variable dynamic output, you cannot use static prompts. Use this exact formula:

- **1. Architecture & Purpose:** Plain language description of the engine's core function and the technical environment (e.g., dynamic HTML/JS).

- **2. Variable Inputs:** What the user provides (dropdowns, text areas, sliding scales, conditional logic fields).
- **3. Dynamic Processing:** What logic runs on the inputs behind the scenes (scoring algorithms, real-time categorisation, matrix plotting).
- **4. Variable Dynamic Outputs:** How the engine responds in real-time (auto-sorting tables, visual charts, live text generation based on selected patterns).
- **5. Export & Distribution:** The exact delivery mechanism for the final output (e.g., 'Download as PDF' or 'Email to yourself').

4.2 Example Prompts From My Live Experiences

The User Story Slicer Engine

"Architect a dynamic HTML/JS engine where users input an oversized user story. Render an interactive UI offering 6 advanced splitting patterns (workflow steps, business rules, data variations, operations, performance, platforms). Upon pattern selection, trigger a variable dynamic output that auto-generates 3-5 perfectly sliced micro-stories, each complete with contextual Given-When-Then acceptance criteria. The UI must include action buttons to instantly 'Download as PDF' or 'Email to Yourself' as a formatted delivery brief."

Dynamic Stakeholder Mapping Engine

"Build an interactive Mendelow's Grid live experience. Users input variable stakeholder data, rating Power (1-5) and Interest (1-5). The engine must dynamically plot these coordinates onto a live 2x2 matrix. Based on the quadrant (Manage Closely, Keep Satisfied, Keep Informed, Monitor), generate a variable, tailored communication strategy for each stakeholder. The final output must be presentation-ready, featuring a 'Download as PDF' function and an 'Email to Stakeholder/Self' integration for immediate distribution."

RICE Prioritisation & Analytics Engine

"Develop a dynamic RICE scoring engine for feature prioritisation. Users input parameters for Reach, Impact, Confidence, and Effort. The engine must calculate the RICE score in real-time, generating a dynamic output table that auto-sorts features by priority, alongside a live visual bar chart comparison. Implement an executive export module allowing the user to 'Download Full Report as PDF' or trigger an 'Email Summary to Yourself' for immediate backlog planning."

4.3 Key Principles of Advanced AI Instruction

- **Demand Dynamic Responses:** Never accept static text when you can prompt for an interactive engine. Ask for variable outputs that change based on user input.
- **Be specific, not vague:** "Make it fast" means nothing. "Load dynamic matrix results in under 200ms without page refresh" is testable.
- **Include complex edge cases:** What happens with empty inputs? What if the user inputs 100 stakeholders? Establish fallback states and clear error handling before the AI writes a line of code.

- **Reference real frameworks:** "Based on Mendelow's Grid" or "Utilize the Kano Model" gives AI a strict, professional foundation to build upon.

5 Testing Checklist for Dynamic Output

Show it to 3-5 colleagues. Do not explain it. Just watch. When testing complex engines, observation is your only honest metric.

5.1 Observation Checklist

- Do they understand the purpose without explanation?
- Can they complete the primary task without getting stuck in a loop?
- Does the variable dynamic output update exactly as they expected when they change their inputs?
- Do they actually use it or just say 'interesting'?
- What is the first thing they try to do? *(Reveals their operational mental model)*.
- Where do they hesitate? *(Reveals UI/UX friction points)*.
- Do they successfully trigger the 'Download as PDF' or 'Email to Yourself' functions? *(Reveals the true delivery value)*.

5.2 After Testing

Take notes during the session. Do not rely on memory. Do not defend your choices. User confusion is an architectural problem, not a user problem. Fix the biggest friction point first. One iteration, one fix. Test again. Iterate based on real, live feedback, not assumptions.

5.3 Quality Criteria for Live Experiences

Check	Standard	Why It Matters
Mobile UX	Responsive on phone	Delivery practitioners will try it on their phone between meetings.
Empty Input	Graceful handling, clear error messages	No crashes ever. The engine must survive bad data.
Export Utility	PDF, Email routing, CSV	The experience is useless if the user cannot take the output back to their desk.
Speed	Under 2 seconds rendering	Anything longer and operational users leave.
Accessible	Keyboard nav, screen readers, contrast	Professional inclusion is not optional.

6 Ship / Kill Criteria

No attachment to ideas. Attachment to outcomes. Killing bad ideas fast is a feature of strong product ownership, not a bug.

SHIP IF:	**KILL IF:**
Solves the core problem dynamically (not a static side effect).	Does not solve the core pain point (even if the code is technically impressive).
People actually use the export/email functions to share results.	Too complex for the operational value it provides.
Feedback is positive and specific (e.g., "I emailed the PDF to my Scrum Master yesterday").	Better, faster solutions already exist (do not reinvent the wheel).
Maintenance is manageable (runs efficiently).	Requires ongoing database maintenance you cannot provide.
You would use it yourself in a live sprint planning session.	Users need extensive explanation to understand the dynamic output.

6.1 Projects I Killed - And Why

- **Backlog Health Analyzer:** Too subjective. What counts as "healthy" varies by team, by project, and by context. No universal dynamic scoring rubric worked reliably across different operational environments.
- **Automated Meeting Scheduler:** Too many edge cases. Calendar conflicts, timezone handling, manager overrides, recurring vs. one-time—the logic complexity completely outpaced the actual delivery value.
- **Requirements Quality Scorer:** People gamed the metrics. Teams optimised for the automated score instead of for actual requirement quality. The engine incentivised the wrong operational behavior.

Each of these took 1-2 days to architect and test. That is the advantage of rapid prototyping—the cost of failure is low enough that you can afford to fail fast, harvest the intelligence, and learn.

7 Real Examples From My Work

The patterns in this guide come directly from live delivery work. Below are examples of shipped, highly-dynamic functionality that made the cut and are currently running on catalystagile.com.

1. **Ignite Intelligence v15:** Architected a multi-agent autonomous operations platform — multi specialist agents, real-time telemetry, three-tier autonomy, and an automated chairman briefing every morning. In production daily use. Built 1 hour a day for 6 months, from first principles. The discipline was Business Analyst / Product Owner; the output is AI infrastructure.

2. **Dynamic User Story Slicer:** Sprint planning wasted 4 hours on story-splitting debates. *Result: An engine that instantly renders smaller stories with Given-When-Then criteria, fully exportable to PDF. Used across multiple teams.*
3. **Stakeholder Map Generator:** A massive volume of stakeholders on a banking merger, unclear communication strategy. *Result: A live 2x2 dynamic grid plotting system that auto-generates communication plans, exportable via email for high-value project distribution.*
4. **CatalystAgile.com:** Portfolio needs to prove AI-augmented execution capability. Result: 16 pages, 54 live experiences, 9 Ignite Intelligence renders (system boot, Olympus strategic command, Jarvis agent registry, the Bridge routines, Growth, Operations, Chairman's Briefing, Treasury, Three-Tier Autonomy). 8,000+ words of copy, completely architected and written by me.
5. **RICE Prioritisation Analytics:** Feature prioritisation driven by opinions, not data. *Result: A live-sorting engine comparing up to 10 features with dynamic bar charts and presentation-ready PDF exports.*
6. **Acceptance Criteria Engine:** Vague requirements leading to misaligned deliverables. *Result: An interactive questionnaire that transforms ambiguous requirements into testable, strict Given-When-Then criteria.*
7. **Diplomatic Email Rewriter:** Diplomatic emails took 2 hours to draft in high-stress environments. *Result: An engine with variable tone analysis (firm, empathetic, escalation) that instantly formats and allows you to 'Email to Yourself' for immediate sending.*
8. **Sprint Velocity Forecaster:** No easy way to track velocity trends. *Result: An engine generating visual velocity trends, rolling averages, and dynamic completion date forecasting.*
9. **Definition of Done Engine:** Inconsistent quality standards across teams. *Result: Shareable DoD dynamic checklists that teams can interact with, agree on, and export.*
10. **Live Risk Radar:** Risk assessments were static documents nobody revisited. *Result: A dynamic probability/impact matrix with visual radar displays, exportable for immediate executive stakeholder reporting.*
11. **Kano Model Prioritiser:** Teams confusing must-have features with nice-to-have features. *Result: A survey methodology engine with automatic classification based on customer satisfaction impact.*
12. **Business Analyst / Product Owner Skills Assessment:** No structured way to identify capability gaps. *Result: An interactive radar chart across 10 Business Analyst and Product Owner capability dimensions that generates personalized, downloadable recommendations.*

8 Common Pitfalls & Failure Modes

Most AI output fails in predictable ways. Here are the 8 mistakes I see constantly and how to avoid each one.

- **Building before understanding:** Spend more time on Problem Identification than on any other step. If you cannot explain it in one sentence, stop building.
- **Trying to build enterprise software:** You are not building the next Jira. Keep scope small. One input, one output, one purpose.

- **Not testing with real users:** Your opinion of your live experience does not matter. If they do not use it unprompted after the demo, it is not solving their problem.
- **Over-engineering the first version:** Ship ugly. Ship simple. Ship fast. You cannot get back the week you spent on features nobody wanted.
- **Getting attached to your idea:** Sunk cost is real. If testing shows it does not work, kill it. The 1-2 days you invested taught you something. That knowledge transfers to the next idea.
- **Writing vague prompts:** 'Build me a project management tracker' equals garbage. 'Build an HTML form with 3 input fields for task name, priority dropdown, and due date' equals useful.
- **Skipping the Kill decision:** Not everything should ship. The discipline to say 'this does not work' is more valuable than the discipline to ship.
- **Forgetting about maintenance:** If you build dozens of live experiences, you maintain dozens of live experiences. Static HTML/JS equals zero maintenance. That is a feature.

9 Infrastructure & Costs

You do not need an enterprise budget to execute this. The full stack costs roughly $60/month. The work, however, is in the business requirements and the architecture – not the bill. Traditional development of Ignite Intelligence alone would cost $80,000+.

9.1 The Stack

- **Claude Code:** Primary building tool — where most of the 54 live experiences, Ignite Intelligence v15, Ignite IDE, Ignite AI, and this website were built.
- **Claude CLI:** Command-line interface for rapid iteration. Useful for quick changes and debugging.

9.2 The Division of Labor

AI is an enabler. Like Excel is an enabler for financial analysts. Like Figma is an enabler for designers. AI is the prototyping engine for Product Owners.

What AI Does	What I Do
Writes HTML, CSS, JavaScript.	Decide what features to build.
Builds live experiences, dashboards, calculators.	Prioritise competing needs.
Makes changes when I describe them.	Understand business strategy.
Removes technical barrier, generates code in minutes.	Know when to say "no".
Fixes bugs I identify.	Test with real users and make product decisions.

10 Your Turn: The One-Week Experiment

Total time: 4 hours spread across 5 days. That is all it takes to go from an idea to a shipped, usable solution.

- **Day 1: Identify.** Identify one repetitive problem you face as a Business Analyst or Product Owner. Write it in one sentence using the problem statement template.
- **Day 2: Build.** Spend 1 hour with your preferred AI platform. Describe what you want the same way you would write a user story—with inputs, outputs, and processing.
- **Day 3: Test.** Test it with one colleague. Do not explain it. Watch what they do. Take notes. Where do they hesitate?
- **Day 4: Iterate.** Fix the biggest confusion point. One fix only. Not ten. Test again.
- **Day 5: Decide.** Ship or Kill. If it works, share it with your team. If it does not, try a different problem next week. Either outcome is a win.

You do not need to be a developer. You need to know what problem to solve. Describe it clearly. Test and iterate. Make strategic decisions.

That is Product Ownership.

READY TO START?

Try it this week. One problem. One execution. Five days.

Explore the **54 live experiences** at catalystagile.com
All free. No signup. No data collection.

Connect directly: ignatious@catalystagile.com

CatalystAgile

I turn operating need into shipped product — BRD-precision specifications, AI-augmented execution.