

# Target User Classes

## Target User Classes

### Competency Center Head

*I'm sitting on a pile of knowledge and a bunch of experts. I'm compelled to turn it into money. I want knowledge collected, I want to know my experts, I want my top contributors known and retained, I want to use knowledge & experts to solutionise & drive RFPs to close fast.*

### Pains

1. We know things, but I don't remember what we know and who knows it.
2. People leave → knowledge leaves.
3. People I have — I need to justify compensation for them.
4. Respond to RFP: draft an answer using knowledge from the Playbook.

### Experiences I Seek

1. **Fast.** I want <this> to take minutes, not days.
2. **Cognitive offload.** I can name the names and I know what needs to be done — I just want to cobble an outline and hand it over for someone to complete.
3. **Polished.** I want investor-grade presentations & diagrams.

### Social / Emotional / Functional Progress I Want

1. Show my folks I can add value with AI.
2. Finally know who contributed what.
3. Showcase our expertise via LinkedIn.

### Jobs to Be Done

1. **Expertise by tag** — show me my experts: I need to pick someone for a particular job.

2. **Contribution review** — evaluate contributions over the quarter/year: who are my best knowledge sharers?
  3. **Reference data** — how long it takes, how many K tokens will be used, etc.
  4. **Blueprint draft** — given the drivers, here is how we can build it and who is available to do it.
  5. **RFP response** — here is what needs to be → how to cook it → what we will need → how much it will cost.
  6. **Refresher** — “In my time it was like this: ... Is it still so?”
  7. **Level Up** — “What is the Delta Lake? How do I build one in AWS?”
- 

## Pain Relievers

1. **Expertise Discovery playbook** — run with Outlook MCP against your inbox → get proposed Playbooks & Workflows & expert assignments in Jira to create it using “Playbook of Playbooks”.

## Gain Creators

1. RFP playbook.
2. Expertise Discovery playbook.

## Value Proposition

1. Retain organizational knowledge & capability.
2. Know your experts & measure their contribution.
3. **RFP Playbook** — turn Playbooks into RFP answer drafts in under an hour.

## Junior Engineer

*I just joined the project and I'm required to follow TDD and write provable, defensible and observable code (what the heck does that even mean?..) I struggle to learn all this stuff and I'm constantly not sure that I'm doing fine - I'd love someone (person or AI, as AI is less embarrassing to ask stupid questions) to sit with me and talk me through what needs to be done. I want to worry less and be more confident at what I do. Maybe I even want to grow professionally and learn advanced techniques.*

## Pains

1. I don't know where to start.
2. Is there a checklist I can follow?
3. I don't understand particular “hows” (think “this is the first time I see list comprehension with lambda”) or “whys” (think “what is the barrier between producer and consumer and why do we need to wait by the barrier”)

## Experiences I Seek

1. **Simple.** I want all advanced concepts explained in a plain language.
2. **2nd pair of eyes.** Look at why I do, warn me if I'm attempting something stupid.
3. **Guide & coach.** Guide me step by step, explaining along the way.

## Social / Emotional / Functional Progress I Want

1. I don't want to look stupid.
2. I want to feel confident.
3. I want guardrails so I can tell "its not my fault - I did everything by the book."

## Jobs to Be Done

1. **Guide** — walk me through what needs to be done.
2. **Lead pilot** - do things on my behalf, explaining me as we go.
3. **Checklist** - verify what I did, clean any tails I may have left behind.

## Ideas in that regard

1. Steve[McConnell] agent: explains everything as we go, suggests recommended reading.
2. "interns" tag - when workflow discovers it invokes Steve agent, who explains as we go.

## Lead Engineer

*I need to task my team supplying them with the high quality task definition to minimize their chances to screw up. At the same time, I don't want to spend time detailing (thus if someone did it and I just review - this is fine). Also when they done - I'm supposed to review what they do (especially interns) and give them feedback on how to do better with the design & practices to grow their craft, and its annoying - I need to read a lot, think a lot, write a lot, and also I feel emotionally insecure providing feedback because I don't know how and I'm not sure I'm good enough to and I'm afraid my message will be ignored.*

## Pains

1. Writing detailed task briefs is the only way to prevent juniors from going sideways — and it takes forever.
2. My standards live in my head. Every new developer, every new AI session — I explain the same things from scratch.
3. Giving feedback is emotionally expensive: I read a lot, think a lot, write a lot, and it still might be ignored.
4. I'm not always confident enough to give hard feedback — I'm afraid it comes across as opinion, not craft.

5. I can't be in every PR, every chat, every standup. My know-how doesn't scale.

## Experiences I Seek

1. **Delegation confidence.** I hand off knowing the brief is solid and the AI will enforce my standards.
2. **Leverage.** My craft applied consistently, even when I'm not in the room.
3. **Feedback without friction.** I reference the playbook/standards/BDFLs/experts, not my gut — it's easier to give and easier to receive.

## Social / Emotional / Functional Progress I Want

1. Be the person who raised the bar — without being the bottleneck.
2. Give feedback anchored in shared standards, not personal preference.
3. Have my craft outlive my tenure on the project.
4. Less cognitive energy spent.

## Jobs to Be Done

1. **Task brief generation** — given a feature spec, produce an issue with context map, do-not-do list, and acceptance checkpoints already written.
2. **Standards enforcement** — AI reviews code against playbook rules, not just linting; flags violations with a reference to the relevant activity.
3. **Onboarding** — new developer joins, reads the playbook, knows how we work. No 1:1 walkthrough needed.
4. **Feedback anchored in craft** — “per Activity BPE-02, services must not contain MCP-specific logic” is easier to write and harder to ignore than “I don't like this.”
5. **Preserve architecture decisions** — the reasoning lives in the playbook, not in my memory or a Slack thread from 2023.
6. **Evolve standards** — when I find a better pattern, I submit a PIP. It gets reviewed, versioned, and the whole team picks it up next sprint.

## Pain Relievers

1. **Activity guidance + Rules** — task brief template is already written; AI generates the issue from it.
2. **Playbook-anchored feedback** — “the playbook says X” replaces “I think you should X”; emotionally safer for both sides.
3. **MCP reads the playbook** — consistent output regardless of who on the team prompted.

## Gain Creators

1. Your know-how scales without you — encoded once, applied by every developer and every AI session.
2. Feedback becomes coaching, not judgment.

3. PIPs turn your improvements into versioned team standards instead of forgotten Slack messages.

## Value Proposition

1. Your standards, applied consistently — by every developer and every AI, every session.
2. Task definition that takes minutes, not the morning.
3. Pull request with MEANINGFUL feedback in 15 mins, not entire afternoon.
4. Feedback anchored in shared craft, not personal opinion.

## Quick ideas:

1. PR Review Workflow (can be combined with Steve Agent.)
- 

## Scrum Master

*I have to lay out what needs to be done - next milestone and its tasks, but I struggle to maintain consistency of the documentation and frankly - quite often I feel unsure what are [infrastructure/cross-cutting architectural concerns etc] prerequisites or dependencies because I'm not a specialist, so I'm afraid we will overlook something, and at the same time I'm afraid to ask questions because people will see that I don't know the subject. When the sprint starts - I struggle to understand if things are going fine even when codebase increments are pushed daily (how do I get stuff from git? how do I interpret what I see in git?).*

## Pains

1. Milestone planning takes forever and I'm never sure I got the dependencies right — I'm not the domain expert.
2. I'm afraid to ask clarifying questions in front of the team. It exposes me.
3. Documentation is inconsistent sprint to sprint — different people write issues differently, nothing is comparable.
4. Once the sprint starts, I can't tell if we're on track. Git is a black box to me.
5. When something slips, I find out at the end — not when there was still time to act.

## Experiences I Seek

1. **Coverage confidence.** I want to know I haven't missed a prerequisite or a dependency before the sprint starts.
2. **Invisible floor.** I want the documentation to be consistent without me having to enforce it manually.

3. **Plain-language progress.** Tell me what's happening in the sprint in terms I understand — not commit hashes.

## **Social / Emotional / Functional Progress I Want**

1. Run the sprint without feeling like I'm faking it in front of engineers.
2. Spot blockers and slippage early — before they become my problem at the demo.
3. Be taken seriously as a planner, not just a calendar manager.

## **Jobs to Be Done**

1. **Milestone decomposition** — given a goal, produce a consistent set of issues with prerequisites, dependencies, and acceptance checkpoints already filled in.
  2. **Dependency check** — before the sprint starts, flag what's missing or out of order; surface cross-cutting concerns I wouldn't know to ask about.
  3. **Sprint health in plain English** — translate what's happening in GitHub/Jira into “on track / at risk / blocked” without me needing to read code.
  4. **Consistency enforcement** — every issue follows the same structure, so I can compare across sprints and people.
  5. **Early warning** — alert me when something is slipping while there's still time to adjust, not after the fact.
- 

## **Pain Relievers**

1. **BPE workflow in Mimir** — milestone decomposition follows the same playbook every time; issues come out with context maps, do-not-do lists, and checkpoints already written.
2. **Iteration protocol** — status labels (status-queued → status-in-progress → status-done) give plain-language sprint state without reading git.
3. **Dependency structure in activities** — prerequisites and cross-cutting concerns are encoded in the playbook, not in someone's head.

## **Gain Creators**

1. Sprint planning that produces consistent, complete issues — without domain expertise required to write them.
2. Progress visibility in terms a non-engineer can act on.
3. Early drift detection before it becomes a missed deadline.

## **Value Proposition**

1. Plan the sprint without being the weakest person in the room.
2. Issues that are consistent, complete, and comparable — every sprint, every team.

3. Know the sprint is healthy or at risk before the team tells you it's too late.

---

## Project  
Manager

*> I have a  
roadmap and I  
have to push  
through X  
storypoints  
every sprint  
with 0 defects  
leaking to prod.  
But there is  
always  
SOMETHING  
preventing us  
from getting  
there. I try to  
control certain  
variables - size  
of the daily  
increment, open  
bug count,  
feature cycle  
time, PR  
turnaround time  
etc. but there  
are just too  
many for me to  
assess. I'd love  
a "project  
administrator",  
mini-PMO -  
collects the  
data, does quick  
assessment,  
focuses my  
attention: "OK,  
throughput and  
quality are OK,  
but PRs now  
take 2x time to  
get into the  
main. Seems like  
one of primary  
reviewers is  
down and it  
feels - see, I told  
you we need  
more reviewers.  
It seems @jdoe  
and @markj are  
already doing*

*work which  
qualified them  
for the role.”*

#### ### Pains

1. Too many variables to track simultaneously — by the time I notice something is wrong, the sprint is already at risk. 2. I don't know which signal to trust or which to ignore. Everything looks urgent. 3. The data is in GitLab, Jira, and Slack — I'd have to be three people to read it all. 4. I can see the output (velocity, bug count) but not the leading indicators — the things that predict a miss before it happens. 5. Decisions I make feel like gut calls because I can't present the analysis that backs them up.

#### ### Experiences I Seek

1. **Focus.** Don't show me everything — show me the one thing I need to act on right now. 2. **Evidence.** When I escalate or make a call, I



want data I can  
point to, not just  
my read of the  
room. 3. **Early.**  
Tell me when  
we're drifting  
while there's  
still time to  
correct — not at  
the  
retrospective.

### Social /  
Emotional /  
Functional  
Progress I Want

1. Make  
decisions I can  
defend — with  
evidence, not  
just instinct. 2.  
Be seen as  
someone who  
sees around  
corners, not  
someone who  
reacts to fires. 3.  
Know my team  
is running at  
capacity, not  
burning out or  
coasting.

### Jobs to Be  
Done

1. **Daily SitRep**  
— one morning  
read:  
throughput,  
cycle time, PR  
turnaround,  
open bugs.  
Anomalies  
flagged, not  
buried in a  
dashboard. 2.

**Leading  
indicator watch**  
— catch drift  
before it  
becomes a miss:  
PR queue  
growing, one  
reviewer  
becoming a

bottleneck,  
commit  
frequency  
dropping. 3.  
**Decision brief**  
— when I need  
to act, give me  
the analysis  
behind the  
recommendation  
so I can present  
it, not just  
accept it. 4.  
**Contributor**  
**insight** — who  
is doing what,  
who is  
overloaded, who  
has earned a  
new  
responsibility. 5.  
**Roadmap-to-**  
**sprint**  
**traceability** —  
is what the team  
is doing this  
sprint actually  
connected to the  
roadmap goal?

---

## Pain Relievers

1. **Huginn** ([~/GitHub/huginn](https://github.com/huginn)) — Human-AI OODA composite for engineering PMs. Ingests GitLab and Jira signals every 15 minutes, computes Master Variables (TRANSPARENCY, THROUGHPUT, CYCLE TIME, REWORK, QUALITY, COMPLEXITY, CONTRIBUTION), and generates a daily SitRep via Gjallarhorn AI. Semi-Autonomous mode: Gjallarhorn proposes Decisions, PM approves. Autonomous mode: Gjallarhorn executes.
2. **“Manage Project” workflow in Mimir** (*planned*) — the playbook Huginn runs against: defines what good looks like for each Master Variable, thresholds for escalation, decision authority model.

## Gain Creators

1. Morning SitRep in plain English — anomalies first, context second, recommendations third.
2. Decisions backed by computed evidence, not gut feel.
3. The composite gets sharper sprint over sprint as the “Manage Project” playbook is refined via PIPs.

## **Value Proposition**

1. A mini-PMO that works overnight so you don't have to.
2. Act on the right signal at the right time — not everything, not too late.
3. Decisions you can defend because the analysis is already written.