Stop babysitting your agents... — Brandon Waselnuk, Unblocked

18 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=BIG2SSIBKGC
https://www.youtube.com/watch?v=BiG2ssibKGc

SUMMARY

Brandon from Unblocked discusses the importance of context engines in AI-driven development, emphasizing the need for agents to have comprehensive context to operate effectively without constant supervision. He outlines common misconceptions about context and shares lessons learned from building a context engine that enhances agent performance.

- *Agents require a context engine to avoid being mere "babysitters" for developers.*
- *Three myths about context include: naive retrieval of documents doesn't equate to understanding, simply connecting multiple data sources isn't sufficient, and larger context windows do not enhance reasoning capabilities.*
- *A context engine should integrate static and dynamic data, enabling agents to access relevant information at runtime.*
- *Conflict resolution and understanding organizational nuances are crucial for effective agent performance.*
- *The context engine should respect data governance and permissions, ensuring secure access to information.*
- *Key differentiators of a context engine include unified system context, targeted retrieval, and token optimization.*
- *Real-world applications include automating responses in support channels and enhancing coding efficiency through better planning and execution.*
- *The presentation includes a demo of a social graph tool that helps visualize team dynamics and expertise, aiding in context retrieval for agents.*

Hello everybody. It's good to see you. I'm Brandon. I work at Unblocked. It's a great place. Uh and my goal is to make it so that you don't have to babysit your agents anymore. Um I'm sure we all have a different take on what that means. What I think of is care and feeding. Basically agents whenever you spawn it by typing claude in your CLI let's assume whatever tool you may use they exist and it's like a brilliant software engineer has just spawned and it knows nothing about what it needs to do. It knows nothing about your org.

It's completely zero context in its head. So typically what happens is people have to move through building that context which we'll go through in the beginning. But first, what I'm going to cover today pretty quickly is three myths about how you can stop babysitting your your agents and then three lessons that we learned the hard way building a context engine at unblocked. So our product basically provides this context for agents and we'll tell you a bit about how we built that techniques to care about and I'll show you a repo we actually constructed at our workshop yesterday that will be open source at the end of the week uh which has one component of what's in a context engine which you can lift for yourself and bring into your org if you'd like. So not long ago, you

were the context engine. If you think about that, when you're writing code, you thought about everything. You knew everything. You figured it all out. You were dealing with that. And now what's weird is you're in a weird state where you are actually the context engine for your agents. So a useful way to think about this is how did you build context when you showed up at a company? So day one, you had probably nothing, but you were really smart. You finished school, I don't know, maybe some self-learning.

Then over time you accumulate context by doing stuff at work, meeting market, meeting your team, being like, "Here's a PR, getting it rejected." All these good things built up a lot of your capabilities. And then finally, you became very good at your job because you asked good questions and you knew how to gather accurate context and shred stuff that wasn't helpful for you. This is where we're at right now. Most people here if you look at the bottom is in the you are the context engine stage because you're either dealing with the early phases of AI which was just fancy autocomplete or you're in an agentic IDE where you're triggering every job. What we see with all these businesses that work with us in their AI adoption is it's usually at a varying level of this. This has been adapted from Basim Eld's work. um if you check him out later, great engineer. But basically, this is the type of ladder that we're dreaming for. And far on the side is like a dreamy future that maybe codeex figured out. I think I didn't see Ryan's talk, unfortunately, but everyone else is kind of trying to get there. In order to move through this, you want to get to the curated context layer. That is typically what a lot of teams are doing by creating static repos. So, static stores of a bunch of context that says key things about their company.

These can include of course cloud MD files, agents MD, that those types of tooling, but usually people start to put a bunch of other key corporate context into an area that agents can access to pull data from. The issue with that is those are static content pieces of course. So someone has to maintain and up them or sorry update them as well as they don't have uh the availability of you know actual raw runtime data. There's just a bunch of information that engineers obviously need that don't go into these static layers that you're starting to see which typically look like a file system. Wow. Context engine.

What this needs to do is basically have all that static content of course but also be able to at runtime when a query comes through from the software engineer typically how do I implement this feature at runtime it's able to pull all that static source across your entire corporate knowledge corpus essentially whether it's any many SAS apps different systems or records and pull in the runtime singles in order to analyze reason across all those surfaces all those different data stores and then run exhaustively to actually find all of the things that are important and then send a token optimized aka small response to the agent with all the details it needs to then execute its next steps. So you'll see through this but typically that means getting the best context up front makes all agent choices and actions after that even better. So if you give it for example a key research packet of like hey I want to do a new integration and it you drop a packet so that it creates a good plan and that information says here's our patterns we use factory pattern we do these things like all the things about your organization it's then able to trigger its background agent jobs to go gp your codebase to do the things it needs to with higher accuracy which means it's more token efficient and it gets the job done faster.

This is the problem from our our friend Kaparthi. The gap is not intelligence at this point. It is context though mythos sounds pretty cool. Um so the problem is people think that access is the answer but it is not

understanding. So providing your agent tools with MCPs with pipes to different sets allows it to access that. But you have to remember you day one when you showed up at work you don't know where things are and you definitely don't know what you don't know. So there's probably some service over there you've never heard of before in your life. You're working on a thing, your agent's like, "Oh, I did it. I wrote the whole code from scratch." And then your senior engineer is like, "Hey, bro, we have a service." And denies you. So, one thing you'll see is we actually triggered this task. That's a real prompt. You'll see data later in this uh slide about the outcomes. We did it with unblocked a context engine, and then without one, but it had an MCP access to each SAS tool that was required to get the job done. And I'll show you the differences.

The short story is you kind of get this. The naive run which just had the MCP access basically passed all the code checks. It compiled but the senior engineer was like this is totally wrong and what it tried to do would have broken our entire system if we had shipped it. So three myths about building context. The first if I do naive rag over my docs that is context. Unfortunately that does not work. Naive rag picks a bunch of things. is it puts a data store there and the agent can then crawl across that data store but it typically falls down because there's something known as satisfaction of search. This is a known phenomenon in radiology but the short story is if you get an X-ray because let's say you have a lung problem the radiologist will scan and when they find something they're like oh there's something on your lung they stop looking because they think they found the answer to the symptoms you have told them and this is very bad in medical health of course because there may be other things wrong. So what happens with an agent is if you say make a zenesk integration it will go it might call an MCP and the first piece of data it finds it goes oh this this must be the pattern it stops looking so the issue is if it's not exhaustive it will not find the actual root cause or it may not find the correct best way to implement and just a bunch of other problems can happen and basically by the time the agent output is done you read it as an engineer and go no and then you're in a doom loop where you're like let me correct you it's actually over here I'm going to point a file and so you're babysitting If I just connect enough MCPs, I'm done.

I think I've spoken to that. They're there. They're pipes. That's great. But they don't provide understanding or reasoning across it. And then finally, we did think this for a while. We dreamed of the 1 million context window. It's here. I don't know if anyone's ever whacked it full with something and then tried to get the agent to do anything. It can't. Um, it basically just can't reason over that much data. It's just not super helpful. It just sits there.

there's no entities and relationships and there's all these things that we need for these agents to be most effective. Um, so the bigger context window does not solve it. There's a bunch of compute reasons why even if we got to 100 million in a context window, it's still not going to help other than needle and haystack problems if you're obviously like fine fine waldo. So basically this is what we see the classic waterfall code that compiles is what the agent can see. But typically today, they miss all of this because it can't see it. It doesn't know if it's there. It would have to run for so long grepping in a session to actually get your factory patterns or other things across your codebase that you'd burn a bajillion tokens and then when you close that terminal window, bye-bye. You got to just do it again. And no one wants to repeat this cost.

So this is why you need a context engine. It understands who you are and what information actually matters. So

a key component of this is a social graph because you use that as a pivot point because if I ask how to do the Zenesk integration, the context engine should know which code bases I work in, where my PR history is, who I work with, and what I mean when I say that because at a large or we deal with companies that have 20,000 members at our that are customers of ours, it's very different.

So you need to be able to reason that. And by the way, a context graph is an incredibly useful technique for building these things. We'll talk about it. It should resolve conflicts. I don't know how many times I've looked at source code that's running in Maine and we go yeah that's the source of truth but there's a Slack conversation where the CTO says that was implemented wrong which is right a context engine must be able to settle that debate and by the way a a graph of social graph helps with that because if you see the CTO saying in the Slack thread that's wrong the CTO is probably right. So the context engine reasons about that and goes well the code says this, Slack says this, that's the CTO. We should probably tell the agent what the CTO said and of course provided the source code etc. So it passes what's truth. So it handles truthiness. That's a tough problem. We have a lot of techniques in our product to solve it. It is not fully solved. It should respect permissions and governments. This is pretty basic. It's one of the reasons this is delivered over MCP is you can carry the OOTH model through for data governance and a bunch of other reasons that matter in scaling businesses. I mean, as soon as you're 20 plus, typically this matters because some data should not be accessible to others. So, when you build your engine, you do not want to put everything in there, especially when you think about we ingest Slack conversations and Microsoft Teams convos. So, if it's you, we will return responses from private chats, but if someone else asks a question, we will never show them private chats that aren't theirs. That's just one easy way to think about it. And it should deliver the right context, the right model at the right time in a token optimized way. This is how ours works.

So the short is we ingest a bunch of data sources. It sits in our engine. We have six key differentiators which I'll go into in a sec. And then on the output side, there should be many surface areas where agents and humans are able to interact with the context engine. One of ours is simple. Human engineers in Slack just chat with it and ask it questions all the time to get data they need. I'll show you an example. Um, but then of course agents as you move to background agents, they need a context engine in order to run headlessly or to run in the background or run in the cloud because they have to be able to ask questions of a machine, not a person because otherwise you're not going to wake up to a PR. You're going to wake up to a am I allowed to use this tool? And that's not helpful. So these are the six. We're going to move at pace. But the short is I've talked to a lot of these, but these are kind of marketing terms. Unified system context. So it should be able to reason across all of your systems of records. It has to be able to do targeted retrieval.

Conflict resolution as described there are many times where the docs and this and that are conflicting. So how do you settle that? That data governance problem. So secure access model, personalized relevance, building social graphs, knowing who you are, who you work with. And finally, of course, token optimization. This is becoming a pretty big issue. A lot of benefits you get on token optimization is just by having a context engine because you don't rerun those GPS for the agent to know. But also with an engine, if you reason across everything, it's then able to compress the response into exactly what the agent needs and only send that back as an answer.

That task I talked about, I'm not going to lie, we asked Claude to do the comparison, so it made these bar charts and numbers, but it did pull out all the key points of this. That same prompt, one was naive. It had all the MCPS it needed to get the data and the other had our context engine only. This is the difference across key principles of engineering. But these are funny. It like didn't catch that we use bedrock as a a fallback. It shipped like bugs.

There was one that broke the custom callers. The short story is if you're working at any form of scale again 20 plus agents are just going to try to mock things and it'll look like prototype. It's not mergeable. if you get a context engine. When I put up this PR, our senior engineer for the one with the engine basically gave me a nitpick and was like, "Yep, you can merge this. Just just fix that." Great. These are some of our hard lessons.

We did try to optimize for access, not understand. We like the bottle will handle this. Like the agent's totally going to figure it out. It'll collapse into mythos or whatever. It it hasn't. It's been years. So, we were like, "We have to solve this problem another way." And I think that's correct based on actually Anthropics launched last night with cloud agents and Ryan's talk this morning. You have to get context into the harness and an engine is the way to do that. We hid conflicts instead of servicing them. The agent would actually just pick when we found conflicts at first because we like it can't be that bad. It's that bad. So solving solving conflicts is an important problem. And then finally, this is a really fun one.

We thought as good answers happened, we actually got feedback loops on those. So we were caching them for latency. If you cache a good answer, basically it's like when you write docs, right? The moment you write it, it's no longer valid because things are changing. So if you cache a correct answer and then tomorrow someone asks the same question and you answer it, you you probably lied to them now because things probably changed in a 24-hour clock. So the system is not the same. Obviously, some questions I'm sure are stable, but this led to a lot of problems. So I highly recommend against even if it's optimized.

This is where AI forward teams are. They're using context engines in all of these cases. So I know we're all engineers, but I'm sure that we support others in our orgs like a ask engineering channel. Our context engine is sits in every customer's ask engineering channel. It detects if a question is asked, it scores confidently and then it responds automatically. So when support teams, sales teams, whoever is like, hey, what's running in prod? What's this? The context engine just answers them and deals with the issue just like it would answer an agent asking for data. Um, so there's a lot of ways like that use case I just talked about, but then ticket enrichment, triage, incident management, obviously working with your agents and coding.

These are all great ways that you get tons of leverage out of getting one of these into place. Teams then customize them. So most fork, we have like a cookbook. They take that repo, the cookbooks full of skills. Um, but then you devise your skills with your standard operating procedures. You obviously can build your own workflows, whether you do that as a skill or some other technique. And then of course custom agents. All of these can leverage that same context engine. So you just get a huge amount of leverage.

This is what I'm trying to leave you with. An agent should write code that feels like it was written by someone who's been on your team for years. Like that's just like we should expect that by now. And this is one of those

techniques to get you there. I can do a brief QA and I can give a demo. I have three minutes left. So what's your preference? Demo. >> Cool. So this is the tool I talked about at the workshop. We're going to open source it. This is one component of a key of a social graph. And so when you run it, this will be available.

I think it's Monday. We're going to actually open source closed source right now. There's a whole setup, but we had a bunch of teams hack against it and ship code. But basically, it will build you one of these. Uh I'm going to zoom in. This is our engineering or as you can see there's different nodes and edge. It's a basic graph, but it's a social graph. Rasheen is a goddamn machine. So he that's how much he ships is by the size of the node. If you go look at this is algorithmic procedurally generated.

So you'll see the tool but in short you can see on the right who he's working with. Sorry the screen is like kind of small. Great. Kind of worked. Um so you can see who he works with whose code he reviews what area uh he worked in. We use labeling with an API key from Enthropic. Oh god now it looks terrible. Let's zoom in a little better. Um, sorry, it's a fun tool, but basically this expert graph would allow you to when a query comes across, we go, "Oh, you're Rashine. You work with these people." Great. It'll pivot on that data. It'll then zoom into the code bases that he's dealing with.

So, when you as an engineer ask maybe not the best prompt of all time, but you're like, "Hey, I got to get this done. There's a bug. Got to fix it." It'll know who you are. It'll pivot on that. It'll probably find the bug that's like correct. And again, it'll use this component to do a bunch of things and make decisions as it traverses in order to reason to give the agent the exact answer that it needs. So this this tool like generates this for you. You just point it at your code repo. It'll do a construction. Um but it does things like creates experts across like various areas in the business. So in our libs, our services, I'm quick scrolling, apologies, we got time. You can check a heat map gr of who works with each other, like who reviews what, who authors what. You can check peer tables, you know, who Andre works with, etc. So, this data available in a context engine, very useful. And again, this will be open source. Uh, if I think you got a badge scan, I'll I'll just email you all as soon as it's open source. That's cool.

Another quick demo. Ghosty, my boy. So, in this one, I actually used our MCP and I just straight up said, how do I make a new first class integration to Zenesk? I just said, use the MCP. it would probably have picked it up, but I want to make sure for this demo that it did. It ran it chose to use our research task tool. You can see that it constructed a query. So the agent did that based on the shape of our MCP. So it wrote the right query, ran that. It did effort high for reasoning. It got the data back. Then it triggered its explore agents. This is key because now they're exploring the right place after this research packet came in. Great research results. Did the thing, did the thing, wrote me a plan. So if you look at this plan, you do not know my source. That's fair. But if we just scan it, like it found all the things that matter like registering our provider, obviously we have a factory pattern. Um, I'll just pull through, but like the library modules, client, like this is like one hell of a plan is the short story. And it's like pretty correct. I would probably prompt this a couple more times to get it totally right, but at any time while it's executing, it's able to keep calling our MCP. So typically what we see is use the engine for planning, run execution, and then basically as you get to code review, leverage your engine again because that engine is very good at code review and it's extremely good at planning.

That's all the time I have. So thank you all very much. I appreciate it. Uh I'm at a booth at G16, so come by if you have questions.