Belinda Neal
00:29 - 01:00

Good morning. Good afternoon.

Good evening. Welcome to our webinar today on modernizing legacy code with Claude Code.

We're so delighted you're here. Thanks for spending the time with us, and we really look forward to today's session and demo.

We will have time for questions, which is exciting. First and foremost, we're gonna introduce ourselves, your speakers for today.

Just to introduce myself, my name is Belinda Neal. I'm the managing director of financial services here at Anthropic.

My work sits at the confluence of financial services in our industry and really look forward to today's conversation. Harsh, I'll pass to you for an intro.

Harsh Patel
01:00 - 01:15

Hi, all. Good to meet you here.

My name is Harsh. I'm part of the applied AI team here within Anthropic.

I work very closely within my role with a lot of our financial services firms to help deploy Claude and get the most out of what Claude is able to do. Morgan, off to you.

Morgan Lunt
01:15 - 01:27

Hey there, folks. My name is Morgan.

I work on the Claude code team. I have a special, passion for code modernization and working with large enterprise customers.

So I'm excited to talk about some experiences I've had in that in that space today.

Belinda Neal
01:27 - 03:43

Thanks so much. Okay.

Let's go through our agenda today. Firstly, we're gonna talk about why Modernize Now and why Program Store.

And, obviously, it's a very exciting week for us here at Anthropic with the lease release of our latest model, Opus 5. 5.

Next, we're gonna talk about how agents can transform modernization. Next, we're gonna go through the code modernization plugin and actually go through a live demo.

Next, we're gonna talk about some success stories and getting started, and then we're gonna spend some time on Q&A at the end. So if you wanna start adding your questions to the webinar, feel free to do so, and we'll have time to prioritize those towards the end.

So just some housekeeping as we start. Firstly, a recording of this session will be distributed by email in the next twenty four hours.

So if you have to miss some or you want to watch it again, it'll be available for you. Second, we just covered questions.

Feel free to add them through the course of the the demos and the and the conversation. And then lastly, feedback.

Please do fill in the feedback survey towards the end. We'd love to hear from you, with further questions and and question and ways to engage you in the future.

Okay. So as we start, we wanted to ask the audience a couple of polls.

We have about 2,000 people on the webinar now. So let's just start off with number one.

Have you modernized code with Claude? So let's just take a few minutes to allow people to answer this question. Okay.

Let's put that up, and we'll see what people's responses are. Really interested to see what people come back with.

And Harsh, Morgan, really curious to hear also about your perspectives on customer conversations today. It's certainly something we're seeing a lot in financial services and and hopefully more broadly across the ecosystem.

So let's just give it thirty more seconds. It looks like it's a little bit mixed so far, so it's quite interesting.

Okay. It looks like it's fairly consistent.

So I think the results are pretty mixed, actually. It looks like some folks have actually made some progress here, about 30% in complete or in production.

Twenty two percent of you are underway. That's exciting.

26% haven't done it yet, but it's on your roadmap. That's excellent.

Hopefully, this webinar today will help give you some skills and tools to be able to do that. And lastly, 21% not yet just exploring.

This is the perfect time to start. So, Harsh, Morgan, any any observations or comments based on these results?

Morgan Lunt
03:43 - 04:10

I I would say this. tracks with my conversations with with customers.

I mean, we're kind of at the point now where mid maturity and the ability to do large scale code modernization with Claude. I would say a year ago, this was kind of experimental, kind of trying to hack together and see what you can do.

And today, it's really ready for for production workloads and some serious migrations, which we're starting to see customers have a great degree of success with without as much, hacking around that they would have to do six months one year ago.

Harsh Patel
04:10 - 04:25

Completely. agree.

And, you know, with the pilots that are underway, another thing that we hear very often from customers is once we get started, how do we really industrialize this at scale? Right? So we're gonna be covering some of these elements here in this, presentation.

Belinda Neal
04:25 - 05:11

Excellent. That's a great foundation.

Okay. Let's get going on to our second poll, which is to talk more specifically about, what you've actually focused your efforts on.

So we'll load up the second poll now, which will actually ask you, which of these efforts have you, taken on? So if you just take a second for those of you who have done one or underway, feel free to answer because we're very curious about which areas of focus that you've had so far, and that will also give us some insight about who's in the audience and what you wanna focus on today. So let's give this poll a second and see what folks come back with.

Okay. We'll just give it give it a couple of minutes.

It looks like it's coming in relatively consistent with what we're hearing.

Harsh Patel
05:11 - 05:19

There's a lot of Java or dot net upgrades. This tracks well with, like, what we're hearing from a lot of customers as well.

Belinda Neal
05:19 - 05:33

Excellent. I think documenting legacy code and extracting business rules is definitely foundation, so it's really good to see, the responses there so far at about 30%.

I think that's probably where a lot of people start. Morgan, do you wanna add on to that as well?

Morgan Lunt
05:33 - 05:53

Yeah. I mean, just, we'll touch on this in the demo a little bit, but understanding what your old code does is not always a given, especially for older mainframe style type code.

And so documenting those rules, extracting business logic, figuring out what it does before you go ahead and reimplement it is always gonna be so important. That's that's where you're gonna wanna involve your business stakeholders.

Belinda Neal
05:53 - 06:46

Excellent. Well, thank you very much for responding to that.

I think we've got a good baseline of what we've touched on and hopefully we'll share some things here like, COBOL modernization today that will give, people a little bit more color on those as well. Okay.

Let's go back to the slides. So what are we seeing? We we did a little word cloud here just to give this group some some inspiration around what we're seeing people starting to do with Claude Code with code modernization.

Obviously, there's a large number of specific use cases that we're seeing people test and use so far, and we really wanted to share these as a baseline area to help everyone here think about what would be relevant for their roadmaps and

what they're thinking about so far. So hopefully, we'll get into some of these today.

We obviously won't be able to cover all of them, but again, there's lots of different opportunities to apply Claude Code to your modernization projects. So with that, Harsh, I'll pass to you and we'll get going in terms of the why behind code modernization.

Harsh Patel

06:46 - 12:26

Perfect. Thank you so much, Belinda.

Before we get, into the demo itself, I just wanted to spend a few minutes walking through what we are seeing time and over and over again, whenever we speak with our customers. Many of you folks here in the audience are presumably engineering leaders working in product or what have you.

And the thing that we hear oftentimes is the fact that many of the mission critical systems that your businesses are working off of are powered by code bases that are maybe years, if not decades, old. And so one of the things that has happened over the time as this code has been written is that there has been a lot of institutional knowledge loss as many of the original people who may have built it have long, left the organization or transitioned roles.

What this really means is that how this shows up in terms of your product road maps is that not only does this make it maybe difficult, if not impossible, to be able to ship new products based upon what your customers are demanding, it's also really hard to be able to staff against, you know, legacy frameworks such as COBOL or maybe older frameworks of dot net or anything of that sort, to be able to really rapidly build on top of new capabilities. Lastly, of course, is the fact that many of these mission critical systems may stay exposed, and so there might be a plethora of different vulnerabilities, that are there that might be subject to, security risk.

Now, historically, whenever you might have started these modernization initiatives, in the past, you might have hired or expended many resources in house to be able to either document or extract business logic by hand, which takes quite a bit of time. You would have had to, like, rehydrate, any of these external consultancies with all the information and knowledge.

And the other pieces of large risk is the fact that if you had to do this with, like, one big, rewrite, you might have budgeted maybe years of expertise or time budgeted against that specific task. And what's more is that many of the original tools you might have currently might just, translate, for example, COBOL to Java without paying down any of that technical debt, I e, JOBAL.

So these are many of the challenges that we see time and time over again. But this is precisely what makes agents really, really effective.

You're able to delegate a lot of the volume work to these agents to be able to read, map, document the code using your organizational skills that you might have and be able to prompt it in a style in which that is going to be most conceivable for the initiatives you're working on. In addition, instead of just translating from COBOL to Java and doing a DOBOL translation, you can really build from the ground up using the recovered specs that you might have.

This is precisely where a plot code really accelerates modernization programs. And for those of you who may be new to what Claude Code is, this is a agentic coding tool that lives within our terminal.

Our product and engineering teams work really hard to be able to elicit the best capabilities of Claude. So in the context of these code modernization programs, it is able to efficiently search through large code bases, whether they be hundreds of thousands, if not millions of lines of code long, but then also work under uninterrupted for many hours at a time.

And for many of you who might be coming in from maybe regulated industries or regulated institutions, the best part of this is the fact that your code stays on premise and you're able to get plugged, plugged and ready to go using other Bedrock, Vertex, or our platform. What's more, as Belinda had mentioned at the start here, we just released earlier this week our latest model, Claude Opus 5.

5. What is really exciting is the fact that this performs at the level of Claude Fable 5.

1 for most work while still costing roughly 40 percent less to be able to run compared to Opus 5. As a couple of headline metrics here that, you know, we are referring to within our launch post, one early tester was able to actually migrate roughly 700,000 lines of code in less than one day, showing the power set it has for many of these agentic coding tasks.

But, also, for these long running, you know, problems you might be working on, we reduce the cache reads, to be 60% less, thereby really lowering the cost to be

able to run any of these workloads. At this point in time, you might be asking, well, how do I get started? Right? Morgan Lunt wrote this, code modernization plugin, which is made open source, which he'll be presenting right after this.

But this plug in enforces a workflow where you don't have to actually read any documentation, but rather point it to one of your legacy repos that you have, and you can think of it as actually working you through three high level phases. The first phase is really looking into extracting the business logic that you might have, mapping out the dependencies in a visual format, but then also giving you an assessment as well as linking into maybe your CICD pipelines, your Git repos, what have you, to be able to give you an assessment with respect to whether or not your environment is ready.

At the end of this, it will give you a brief or a bill of materials that you are then able to approve, and then from there, be able to go down one of three paths. Many customers that we speak with either would just want to do, say, for example, an uplift.

So they might have an older version of Java, I e Java 8, go to Java 21, or they might be doing something where they want to do maybe a monolith decomposition, maybe take some monolith and, like, take it into microservices. Or, lastly, if you wanted to actually build from scratch, use the first phase to be able to discover all the business logic and be able to build from a clean slate, using the reimagined phase.

And so with that, I'm gonna pass it off to Morgan who's going to be going through an exciting demo of using this, code modernization plugin, to be able to work through a real live example. Morgan, off to you.

Morgan Lunt
12:26 - 44:59

Alright. Thank you so much, Harsh.

Let me get my screen up for y'all here. Okay.

So you should be able to see my terminal here. Before we jump in, let me just say that this is a plug in that that I made to kind of distill all of my knowledge about code modernization I built up over a couple of years.

I'm not gonna say there's one right way to modernize your code. There most certainly is not.

And I would encourage you to go try this, tweak this, change it, make it your own. It's entirely open source.

We have a link to it, that we're going to share. The idea here, though, is to encode as much as possible best practices for understanding your current code, extracting business rules and behaviors, doing validation after code is transformed, all of this kind of stuff around the actual transformation of code to give you confidence that it's actually gonna behave properly and solve the business problem that that you want to be solved.

You can always just tell Claude, you know, YOLO this from c to Rust. Please make no mistakes.

And, frankly, these days, it's not gonna do a very bad job. But the purpose of this is to show for regulated industries customers, for customers with, kind of a higher degree of trust that they require in their code, the types of steps that you can take to, use Claude in a structured way to do your code modernization project.

And I've tried to encode this in the plug in, make it available for all of you to try to play with, to remix, to change, and and really make it your own and make it work for you. So let's jump in here.

So I am going to execute, my modernization plug in here, and first, let's take a look at what we're working with. So this is a COBOL program, a card demo, which is kind of the de facto standard for open source COBOL.

I'm focusing on COBOL here because I'm very intentionally trying to show a high complexity, very different source to very different target transformation, in this case, going from COBOL to Java. If you're trying to do Java runtime upgrades, dot net runtime upgrades, move from Angular to React, for example, your life is gonna be so much simpler.

And after the COBOL demo, I can show you a very quick Java one. Those are much easier problems to solve.

So we're starting with a really hard one that's gonna take a little more human in the loop interaction just to give you an idea of of how you can go about doing this, and you can strip out steps that you feel are unnecessary for you. Just to give an idea of the, application we're working with here, it's got 44 COBOL programs, 62 copybooks.

We're gonna go through each of these steps here from preflight, understanding kind of your environment, making sure you can actually build the code, you've got all the proper dependencies, on your disk and stuff, assessing what's in there, mapping how the current existing code fits together, what business rules are in there, to make sure that the behavior of the resultant transform code is going to match to what the original one does. We're gonna make a plan.

We're gonna transform it. We're gonna prove it works and show you where we're at now.

One thread to carry through, is this monthly interest job that you're gonna see here. The COBOL code truncates.

It does not round, and so this is a behavior that if we just naively, transform to Java, it's going to be wrong, and I'll show you how we catch that as part of this workflow. So going back over here, we have got our COBOL, and we're loading Claude code and starting the code modernization plugin and telling it that we wanna migrate to Java Spring.

So it's gonna start by executing the preflight, and it's gonna ask me some questions just to try to understand kind of from a human who knows this code base, presumably, hopefully, maybe a little bit, what information the agent may benefit from having that's not kind of preexisting in in the code it's gonna be able to find for itself. So is this the complete system? Are there external dependencies? If there are external dependencies, it's probably gonna want you to, like, try to bring them over on disk so it can look at them if at all possible.

And if that's not possible, you know, it's better for it to know, rather than blindly looking for stuff. It'll wanna know what CI looks like and what your build test, approach is.

Again, with something like Java, this is very straightforward. You can very frequently build your code and and validate it.

For some other languages, this takes longer, and you may wanna take a different approach or batch more changes before you get to verification steps. So we're gonna answer its questions here.

In this case, we do have new COBOL installed on this environment, so it's gonna be able to run some of that original COBOL code to let us do some blue green verification testing towards the end. And asking for bespoke, first party infrastructure and stuff.

Like, if you have a a first party artifactory where you have dependencies that live, it's gonna want to know that and give you a chance to give, credentials or an MCP or whatever you need to pull in, that extra information for it to be able to do its job properly. And then if there's been any prior attempts at modernization, if someone's tried this and failed, if there's partial results, if there's a sticking point that you know, like, man, we hired a consultancy to try and do this, and it fell over at this stage.

And then we tried again four years ago, and it fell over at this stage. That's useful stuff for the agent to know.

So we can try to front load, like, expectations over where difficulty may be, where it may want to extract more knowledge from humans. Again, the more you can throw at it, the the more it's gonna help most likely.

So I've answered the preflight questions here, and it is gonna go to start inspecting the actual files. As part of this, it wants to know, is anything off limits? Like, is this legacy folder, like, something I should absolutely not touch and work in a different directory? Sometimes, but for, IP reasons, there we have customers with requirements like that.

So it's getting to work. And you can see here this, this plugin makes use of a brand new thing we're releasing soon called Claude Mods.

I'm giving all of you kind of a sneak preview into this. The TLDR is that it's a super extensible way to customize kind of anything about the Claude Code harness, including the UI.

So in this case, I've just got, a mod that's implemented through the code modernization plugin to add this nifty sidebar that's gonna keep us on track and show our progress through this code modernization process. So now that it's done the preflight, it's identified for me, okay.

I'm now ready to assess your code, to map it, to extract rules, to build a dependency graph for you, just to kind of flesh out the understanding of this code before we ever get to even thinking about trying to transform it. And so it's gonna recommend it'll give me a preflight report to start with, with my human answers that I gave it here, the scope of what it's supposed to look at, what directories have the code to be transformed, the status here.

So, like, what tools are installed on this machine? It's gonna recommend additional tools if it thinks it's gonna need them. Like, if I didn't have new COBOL

installed here, it was probably gonna recommend that I do so it can try to execute the original COBOL code for validation later on in the process.

It's gonna make sure it understands how to build. The tool chain is present, and, again, it's gonna help you get all this if if you don't have it already.

It's got some information on what the actual program, the COBOL program looks like, the runtimes that it uses, build tool chain, and completeness of source. And in this case, it's noticed, hey.

There's actually some files missing. There are some include statements referencing files that are not on disk.

Probably wanna go try and find those. If you can't find them, it will work around them.

But this is the sort of stuff that is best to identify upfront before your agent just goes and tries to work with, what it's got. So we're ready to go.

What's next? We're gonna start our assessment. So this is where it starts to look very deeply into the code files and see what's up in there.

It's gonna look for security flaws. It's gonna look for compliance issues that it may notice, stuff that you probably wanna be aware of as you actually go through the modernization process to make sure your translated code doesn't have the issues that your original code has.

Again, doing a rote, just kind of dumb one for one transformation, 95% of the time isn't exactly what you want. It also checks for, passwords and secrets in the code that should not be there.

And as previously, it gives me a nice little report because as much as I love the terminal, I think an HTML page is a little more readable. So it's just gonna do that for you.

The agents are really good at this sort of thing now. So we got a bit of a more, complete systems inventory here, all the different files in here, the lines of code, how big they are, the cyclomatic complexity, useful information to quantify the complexity of of this transformation.

It's extracted the architecture. So the code that I gave it had, like, a little read me.

Like, it had very minimal documentation. So it's gonna be inferring from the code itself, the structure, the business domains, the entity relationships, the data flows.

It's inferring all of this from the code. The models can just do this sort of thing now.

Six months ago, a year ago, when I was working on modernization projects like this, you had to build a harness around the models that was very opinionated and very purpose built for a particular type of modernization tasks. You might build one experience for a mainframe modernization.

You might build another experience for Java upgrades and yet another experience for Angular to react, because you had to compensate for deficiencies in the model's capability with this kind of external superstructure, if you will. You don't really have to do that anymore.

The models have gotten good enough that, it's better to treat them in the abstract in in my mind and to tear down scaffolding. What I found is trying to use scaffolding that I used six months ago, leads to worse results at this point because it constrains and boxes in the agent's ability to creatively problem solve and and get solutions to to the problems that it runs into.

So you're probably gonna hear me repeat this this refrain. You could just do stuff now.

Like, the models can just do stuff. Like, don't let your preconceptions from a few months ago, like, impact the way that you think about doing stuff now.

I've been using Fable for a lot of code migration stuff, and it's not even really necessary anymore. I think Opus 5.

5 is giving me kind of the same results as I do this sort of thing, at half the cost or lower. So the cost is coming down.

The barrier entry to the barrier to entry is coming down. Like, you could just do things now.

It's it's very exciting. Back to the demo.

Sorry. So we've got some architecture and data flow that it's extracted here.

And then it's starting to extract business domains and say, like, okay. I believe that these files contribute to these bits of functionality.

Again, this is all, extracted by the agent. And it's gonna start building an understanding of, like, what is this software? What does it do when I go to reimplement it? What do I need to make sure it does? And it's also identified some technical debt in here.

There's plain text path, plain text passwords. There's a broken import export capability.

There's there's lots of stuff that may or may not be be intentional here, and we're gonna have a chance for humans to chime in and tell the agent, like, hey. That, that rounding situation, like, that actually is intentional.

Please do not change that, whereas a blind transformation would not necessarily give you a chance to inject that business context that is not found in code. And, of course, we've got some security findings, unfortunately.

I think, later on, you'll see it also found a PCI violation, which for financial services customers, I imagine, is a bit of an issue. So we're gonna fix all this.

Right? Alright. So now it is taking that dependency graph that it built previously, and building a map of of this code base, and I think this is one of my my favorite parts.

You can see in the sidebar here, each of these little boxes is a size representative bit of COBOL, in the software. And as we work on this, as it reads files, you're just gonna be able to see exactly what it's looking at, what it's working on, what it's transforming, how much has been transformed by by total line of code.

And it has created our topology. So this is something, again, the plug in just does on its own.

You can run this on your own code right now if you want to. And we've got this lovely interactive map of the entire code base of all this co COBOL in the card demo application.

We've got the data stores. I can click into each of these.

It's inferred, the capability, the function of each thing in here and what it does. I can follow every every, connection between any entity, or program in here.

And what I think is really cool, it's extracted business flows and user stories. So the nightly posting when a cardholder's purchases reaches their account, like, how does that happen under the hood? Like that.

You can see that it starts on step one over I don't know where it is. Some place over here.

There's step two at least. And you can follow the exact flow of the function calls of the data and just kind of get an understanding.

Even if you're, like, not ready to modernize your code yet, just having this view and seeing, like, oh, that's how that works can be kind of helpful. It can kinda help detangle in your brain, what you're working with.

You know, if you're got a big ball of mud that you inherited that someone wrote thirty years ago and no one really fully understands it anymore, this is the sort of thing that gives you an understanding that, that's gonna help you going forward. So I love this one.

I think this is really nifty. Moving on, we've got more in-depth architecture diagrams the agent has produced.

This one turned out a little spaghetti. That's a that's a deficiency of mermaid.

Probably could prompt the agent to do a little better here. But, you know, you can actually see the function calls in a in a different way here.

You can see the paths for this nightly posting, and for for other functionalities that that this pro this COBOL application performs. So we're starting to understand kind of the shape of this code, what files talk to what, where data is stored, what the entity relationships look like.

We're starting to kind of form, like, a high level, like, picture, like a map of the Earth that we're that we're looking at here. But that's not gonna be enough to actually transform the code and make sure that it does what we want on on the back end.

In order to do that, we need to have a bit of an understanding of, like, the business rules and, like, what capabilities this software implements that actually matter to a day to day user. And I alluded to this a little bit previously.

It tends to be that the actual business rules that matter for the end user of a piece of software cannot always, with 100% certainty and accuracy, be mined from the code. Even if you have, like, a team of the most cracked software engineers in the world or, like, Fable plus plus plus plus tier models working on it, there's some fundamental context sometimes that does not exist in code that you probably want to try and get from your actual people who are gonna be using this code at the end of the day.

So the agent's gonna take a first pass at identifying and extracting the business rules from the code here. And so you can see it's extracted 323 rules, 41 about calculation, a 146 about, card validation, some life cycle stuff, some policy stuff.

It's found a lot of rules, and it's ranked them in terms of its confidence in them. So some things are just, like, dead simple and dead obvious, and it's gonna say, let's not waste human time trying to verify this.

Like, this is pretty straightforward. Your human time is limited.

My human time is limited. Everyone's human time is limited.

So let's defer everything to the agents that we can and try to leverage our limited human time to be as high leverage as possible and clear up the things that the agent thinks are gonna drive the most value for the the success of this transformation. So we've got a bunch of business rules that's extracted here.

It's prioritized them based on how important it is that a human take a look at them. We can click into a business rule, and it's gonna show me in plain English.

After each authorization, for a known card, the accounts pending authorization summary is refreshed, and the accounts credit and cash limits, blah blah blah. And it's gonna ask me, like, is this right? Here's a suspected defect.

Is that an intended behavior? Is that, like, what you wanted to be doing? And the amount that you engage with these rules is is really up to you. It depends on your use case, the severity of what you're working on.

You can have humans go and audit all of these and hand check each of them and make sure that they are true. You can tell the agent, I don't know.

Do your best. You know, both are perfectly valid approaches.

It really depends on your risk tolerance, on how one to one you need your end result to be. But at the end of the day, the agent is presenting this path to you, and you can make the decision given the the sensitivity of the modernization you're working on.

And then we also have another view where it actually shows you, here's the COBOL. Here's the business rule extracted from that COBOL just to triple check, like, that you're, that you're extracting the intent properly and that it properly understands what what you want to do.

And you can, like, forward these around to other people in your company. You could tell the agent if you got, like, the Slack MCP set up.

Hey. Can you forward business rule blah blah blah to Harsh and because I think he's the domain expert there and make sure he understands it.

And when Harsh replies on Slack, the agent over here is gonna hear it and, write down effectively on disk the the results and keep moving. Again, this is just Claude code, so it's infinitely extendable.

You can talk to whatever other systems that you have via MCP. You can do whatever you want to it.

This whole plug in again is open source. You can make it your own and make it work for your for your business.

So here, I'm just giving it some feedback on some of the business rules. And so now it's gonna put together a brief.

And what a brief is is basically the report that you can take to your boss and say, alright. I think I have a plan for modernizing this, like, pile of mud that we've been dealing with for a while.

Here's how I'm gonna do it. Here's what I'm gonna start with.

Here's how I'm going to validate that it's working properly. Here's my understanding of the business rules and the requirements in inherent in the code that we're beginning with, and here's how we're gonna make sure that, the end result is actually what we want it to be.

Again, you can just YOLO this. I imagine regulated industries are a little averse to that these days, and so this is how you build that additional confidence that you're you're following kind of a prescriptive flow that that is sensible.

So got our brief here. It's citing the the actual files on disk that it's built this brief out of, for auditability and for backwards, for review to make sure that you understand for compliance reasons everything that's gone into modernizing your code, I would highly recommend just making sure that, all of this, directory the agent is working in and the files that it's writing are, are stored in Git.

So you can just have the agent get pushed every time it touches a file or every time it modifies a file, and then you have a very granular high resolution tracking of what file changed at what time, when a decision was made. If someone signed off on a business rule, that's gonna show up in a file change on, in markdown effectively in the agent's working directory.

And if you push this to Git frequently, you can always find exactly at what point in time a change happened and what user authorized that change. So you can be very confident that you can reproducibly explain why the resultant code ended up the way that it did.

And then just to make another thing clear, this is not all just, like, running through, like, inference. Agents are so good at writing their own tooling at this point.

And if I see an opportunity, I'll I'll try to point out an example. For example, generating the dependency graph of all the modules in the code.

It's not just like sending all of the code through an LM to do that. That would be incredibly wasteful and expensive and non deterministic.

It'll write little Python utilities, little bash scripts for itself, and store those away for later so you can go look at them and see, okay. When the agent built its dependency graph previously, here's the Python script that it ran that's a deterministic.

It'll run the same every single time script, and have a really higher degree of confidence that, what the agent is doing is not just kind of ethereal and and temporary. It's reproducible.

So we've got an objective for this pilot that we're going to be proposing in our brief. We've identified a target architecture, and, of course, you can tell the agent that you have different opinions and you wanna do things differently and build on a different stack, do a different architecture, what have you, and it will change that.

It's gonna give a proposed mapping of each legacy component to a new component in the new architecture. Again, in this case, Cobalt to Java, two totally different architectures.

So it's kind of, taking the business rules and then building a sensible architecture from scratch. Got a number of notes here for humans to review, and it's proposing a sequencing.

So especially for very large projects, you probably don't wanna big bang something all at once because if you do that and discover something that went awry in one part of the code base, you're gonna have to go redo the rest of it, and that takes time and that costs money, and your tokens are probably better spent doing more differentiated work than that. So in this case, it's identified a relatively low connectivity bit of the code that does not have a huge amount of dependency on the rest of the code to strangler fig out of the original COBOL transform first, and then with some shims, have it work with the original COBOL and make sure the behavior is is about the same.

So, the pilot is gonna be transforming this one representative unit, and it's gonna explain to us how it how it recommends these phases go, just based on its decomposition of of the code that we're working with here. And to be very specific, it's gonna propose, the nightly posting and intro cycle as the as the module that we begin with.

So without further ado, let's transform some code. I'll approve the plan.

I'll tell it to make a revision to the plan, then I'll approve the plan. Alright.

And it's defined some tests, some rules that it's gonna say need to, pass in order for this pilot to be considered successful and to move to the to the next phase. And let's get it working.

Reading the brief, checking the tool chain. It's gonna do build the original code and the resultant code and do some testing to make sure that the behaviors are the same.

It's writing some additional tests and then finally doing the implementation, which honestly is, like, the quickest part of this whole thing. It does have an architecture critic.

Look at the resultant code just to make sure there's not any anti patterns being, fallen into, and you can give it additional context and and stuff you wanted to avoid there. In this case, because we're going to Java, which is like exposes like a regular web server, it's identified that the proposed implementation has a open rest trigger, to allow an unauthenticated user to start a money moving job.

Very bad. Super bad.

Do not wanna allow that, and the agent caught it for me here. So we can see the, the actual module that is transforming over here, and it's done.

It has transformed, and it has proven to itself equivalence. And so what you can see here is for each rule that it extracted so there's 18 rules that, touch this particular module, the interest calculation module that we talked about.

It'll show me very specifically, here's the COBOL that implements a particular rule, in this case, reading category balances in key order. You can also look at, I don't know, interest is balance times rate divided by 1,200 and then truncated.

So this is where that actual computation happens in code. This is the rule that the agent has extracted from its understanding of what it sees in code.

And here's the Java that implements the same functionality. So you can very simply see, like, before, after, what the agent thinks happens, like, the tests that pin this behavior to characterize that the before and after behavior is the same, and you can have it create this for, you know, the entirety of the code if you want.

Like, it's not that hard. It can do a very good job at this, and build that confidence that the agent is actually doing the right thing.

And this is all just dynamically generated HTML. Like, I gave the agent a little prompting to say, hey.

I wanna see a side by side. I think it's in the the plug in already, so you shouldn't even have to do that.

But, like, they're so good with the flexibility now that if you say, I want this, but I want it to have unicorns and rainbows on it, or I want this, but I wanna have, you know, a fourth column that shows, something else. You can just do that.

There you we're no longer boxed in by, like, what a rigid, purpose built modernization application can give you. You can just have it present however you would like for it to.

And so here's our equivalence proof. So given the same input, both programs, the legacy COBOL one that I ran with new COBOL locally and the Java version, it's run a bunch of test cases against this, a bunch of simulated transactions through COBOL here and identified identical behavior.

It's identified that there's an abnormal end exit code for for some of these applications here, which is which is correct, which is, the desired behavior, and it's identified a fail a potential failure case that it avoided, and it wants to double check that this is the right thing to do. In this case, instead of rounding half up or or truncating, there would be a different result.

This would result in, a different number of cents on user accounts at the end of the day. A naive transformation may run into this pitfall, and you're gonna end up with, like, being pennies off, and your accounting won't work anymore and everyone will be very mad at you and that is a bad thing.

So this is the sort of thing that is encoded into the plug in and into the agent to really look for and pay attention to. And now you can see here, after after that fun little exercise we did, we are up to 2% modernized on this code, which feels like not a ton.

Right? But we had so many opportunities here to make sure that our understanding of what the existing code is supposed to do is right. We have chances to bring in stakeholders from other parts of the business to, align that the end behavior is actually what they want to see and what they, are gonna create success on.

That's the point of this workflow, the point of putting the structured behavior around increasingly intelligent agents. This is intentionally a plug in, not a purpose built product because I expect that as the models become more capable, more and more of the scaffolding is going to to wither away and become unnecessary, and we'll adjust it as as that happens.

But I think there's always gonna be a need to understand this business context from the company, from the people that are gonna be using your piece of software, and this is intended to give you lots of good touch points, that are high leverage in terms of human time to inject that context and that business knowledge and that domain knowledge into your code before you go and produce something and send it off to the world to try. So here we can see we've done a full mapping of of the car demo application, we've transformed a module, and I imagine you've got a pretty good idea of how, we'll do the rest.

So we've got our final topology here. This looks similar to how it did before, but if you go over to this little guy, you can see this one has been rebuilt in Java.

We have a 148 tests that passed that prove it works right. And, as we proceed through the transformation, we'll be able to see how the architecture evolves in in this view here.

One other thing that I think is fun to show all of you is Opus 5. 5, and this is kind of a sidebar for modernization, but I'm gonna make you see it anyway while I have a captive audience, is really good at making 3D scenes now.

So I gave it an MCP to Blender and told it with a single prompt, give me a super cool graphic of a mainframe, like, modernizing and, like, going into the cloud ether, and this was, like, a one shot prompt. You can do you can do fun stuff now.

It's very exciting. I think we've got a little bit of time for me to very quickly run through a much simpler Java application for y'all.

And when I say much simpler, we're looking at a half million, line of code Java application. So I'm gonna run through this very quickly.

The important part that I want to show you is at the end when we actually get to the validation. And one thing you'll notice, if you're very, very perceptive is that as it goes through this, it's gonna do different steps and fewer steps than we did for the mainframe modernization because there's no need to do all that business rule extraction stuff if you're, like, just upgrading Java from Java 8 to 17, which is what we're doing here.

You can shortcut a lot of the process. Like, you can basically just, like, do the upgrade, see what breaks, run a battery of tests, try to do a self debug on on things that don't work, sequence, surface to humans when there's something, like, really questionable.

But for simpler things, again, Angular to React, Python node, Java runtime upgrades, what have you, it can kinda just do it on its own at this point. So here's what I wanted to show.

These are all of the tests in the half million line of code, Java application that I used. I used Jetty, which is like a really old Java web server, built on Java 8 in, like, the nineties, and I'm upgrading it to Java 17.

Every box that you see that is green is a test that just worked properly, a test that was already in the Jetty test suite, after we upgraded it. And when we upgraded it, it bumped the the Palm XML and Maven and stuff, and then it looked for, things that broke, and it went and self debugged and, you know, not terribly different from how you do an agentic code, upgrade in Java, like, six months ago.

But the cool part about this is the agent is so good at conveying what it's doing, where it needs help. You can see here, there are 23 differences.

I can drill into each of them and figure out what is different, why does it matter, is it something that I care about, is this intentional behavior, is this not intentional behavior, and really focus my human time on the actual, like, little nits and little behavioral differences between Java 8 and Java 17 that, that I care about for the end result, and not focus on all the, like, dumb simple stuff that just works if I bump the version number in pom XML. So thank you all for bearing with me.

I'm super excited about what you can do these days in in the code modernization space. The models are only getting better.

Like, you can just do things. It's it's incredible.

I I really recommend you all just play around a little bit. And problems that you thought were unsurmountable, like, six months ago, three months ago, even two months ago.

The the frontier keeps changing, and the things that you can do today are different from the things that you could do yesterday. So play around, experiment, run little pilots, like, try things.

Like, we live in this, like, amazing future right now where software is incredibly fun and the boring stuff, we can just make Claude do it, which is which is really great. I wanna make sure to leave time for questions, so I am going to hand back to our lovely moderators here, if I can find the appropriate window to do so.

Belinda Neal
44:59 - 45:19

Excellent. Thank you, Morgan.

That was a really exciting run through, and I love the way you framed. You can just do things now.

So we have a lot of questions here on the side, that we're gonna tackle, and thank you so much everyone for submitting them. Morgan, why don't you take a look through, and we'll see where we can start.

Morgan Lunt
45:19 - 47:13

Yeah. Sounds great.

I'll just start at the top. What's the plug in you used within your Claude code to show, stage on the right hand side? Yes.

So the code modernization plug in, which I think we've shared links for, is is something that I put together. Again, I'm not saying it's perfect, but it's a starting point.

Feel free to make it your own. That plugin, I augmented a couple of days ago, to leverage something called Anthropic mods, which is launching in the very near future.

So y'all got a a sneak peek at a prerelease thing. I'm gonna push an update to the plug in in the next couple of days that includes that sidebar with that mod, and the mod is just what allows for the actual customization of the UX.

So long story short, check mid next week, and the the plug in that's on GitHub will include that sidebar. It'll just work for you.

Looks like we've got another one. How do you ensure no functionality is lost during the migration, and what guardrails do you put in place to prevent regressions? This is kind of the the mixture of existing tests augmenting with new tests.

You can have Anthropic write new tests, blue green deployments that you can do. The kind of validation is really gonna depend on, like, what your before state is, what your end state is, and, like, how high fidelity you want to be in, in your guarantees of parity.

I've seen folks do anything from, like, running parallel databases and having blue green deployments of modernized applications for weeks and months and identifying any place where they begin to deviate, and really digging into the reason and modifying the code to fix that. I feel, like, super careful.

And if you're a little more less risk averse, then maybe all the tests that are already built into your code are good enough. And if they exhibit the same behaviors on your test suite, you're good to go.

It's kind of a judgment call to make depending on your risk tolerance for for the software you're working with.

Harsh Patel
47:13 - 47:29

Morgan, one of the other common themes that's been coming up quite a bit here is how do we make this plug in maybe more customizable. Right? So, lots of organization has their own skills that encode maybe their coding styles or what have you.

How would you approach this?

Morgan Lunt
47:29 - 48:08

I would install the plug in, and I would tell Claude, hey. My company uses this skill.

We write code in this way. We have these organizational standards.

Please modify the plug in to adhere to those at every step in the process. And the super meta kind of, like, mind messing thing about Claude Code is that Claude Code can write mods or plugins or skills or changes to modify Claude Code.

So it's like, identify something you want to behave differently and just tell it to behave differently and check. it out.

And if it doesn't do what you want, poke it in a different direction and keep poking it until you get it to do what you wanna do.

Harsh Patel
48:08 - 48:25

One of the best practices that I've seen here is, you're able to grab, all the documentation. This is usually, sort of, like, in the format of a dot TXT file, and you can just use that as, like, the input contacts for Claude to be able to do just the things that you had mentioned.

Morgan Lunt
48:25 - 49:37

Yeah. Could the can the plug in be used for any source language? It's the models, again, I'm I'm deferring so much to our research teams, are good enough that I have not come across a language yet that they cannot reasonably handle and deal with.

Like, I thought COBOL was gonna be hard for them. Not really.

Like, I've seen, like, weird esoteric languages that I had not heard of previously be handled relatively well. If you identify a language that Claude Code does not work well with, I would like to know.

I'd actually be very curious, and we could probably influence research to, create some environments and do some training on that to to make everything better. How do you monitor the cost of a migration with the modernization plug in? The the really simple way is just create an API key for this particular modernization project, and then you can just identify, like, hey.

Any usage accrued for this project, run through this API key. You can see how the bill goes.

For, like, a total swag don't hold me to it number, that 500,000 line of code Java upgrade that I showed, I think ended up costing a few $100 in tokens. Nothing crazy.

Harsh Patel
49:37 - 49:55

One of the other things that's been coming up in the questions quite a bit, you walk through how as we're going through this, it's able to extract the business rules. How do you make sure that, like, you're getting all of the business rules like you showed this adversarial review? Can you speak a little bit more about, like, the dynamic workflows that's powering, quite a bit of this?

Morgan Lunt
49:55 - 51:17

Yeah. Thank thank you for bringing that up.

So one thing that, like, lets Anthropic reason with large scale modernizations like this, like, there's tens of hundreds of millions of tokens in in this code base. Right? And our context windows are still about a million on all of our models.

So you can't just load all of the code into into the model naively. Instead, we use a multi agent orchestration, setup that makes a large use of workflows, dynamic workflows.

And dynamic workflows are just a Claude code capability where Claude writes a script, essentially, a deterministic script that kicks off lots of other Claudes that themselves can kick off many other Claudes if if they need to, to fan out and mass to do parallel work very quickly at scale, kinda like a map reduce function, and go do the highly parallelized work, summarize up the chain, summarize summarize back to the main agent that you actually interact with. And these parallel agents themselves, they're still creating durable artifacts.

They're still writing files to disk so that if their work needs to be referenced down the line, that can be done. But as far as the human is concerned, they're dealing with one primary agent.

Doesn't really have context issues because it's able to defer out through dynamic workflows, sub agents, and trees of sub agents to do the kind of rote work, like looking at a JCL file and figuring out what its dependencies are and what it does.

Harsh Patel

51:17 - 51:44

Perfect. I think the next question here was very similar to the the the first answer that you gave was around verification.

Right? So how do we make sure that this AI generated code works properly? Which if I understand, Morgan, the plug in not only walks you through, like, setting up an equivalence harness, we can also use sort of, like, other data that might be available. So if we're migrating a data pipeline, using historical datasets to be able to make sure the inputs and outputs are mapping one to one.

Morgan Lunt

51:44 - 51:57

Yeah. Exactly.

It's the same way you would verify the work from a human consultancy that you hired to do a modernization project. It's you run the thing side by side, you look for gaps, for signals, and you iterate.

Harsh Patel
51:57 - 54:02

Perfect. Alrighty.

I think, we've been getting a lot of the same types of questions. So one of the things maybe we can start, wrapping up with is next steps.

Right? So how do you get started, and what are some additional resources that you might be able to tap into? So there's, many different, things you can do. The first of which is, of course, the plug in here is made open source.

So go ahead and you can just type in this incantation, forward slash plug in, install the code modernization plug in. This is made available for you.

You can also then adapt, for your organization's own needs. Again, this is something that, you know, if you want to augment further, you have the flexibility to do so, and it is language agnostic.

The additional resource here, which is also posted in our docs, is the fact that we do have a field guide. So this is born out of real lived experience that our forward deployed engineers who have been working with customers like yourselves have learned on the field with respect to doing these types of exercises.

So please do give that a look that walks through in a very similar vein These sets of steps that you can think through in terms of setting up the equivalence contracts of, making sure the code is actually behaving the way that you, want it to, as well as setting up, these different types of, harnesses, like the code harness you plug in, to be able to go through that end to end. And furthermore, many of our engineers internally have also worked on many of these, code migration, projects internally.

So feel free to be able to read, one such example, which was taking, a million lines of Zig to Rust, which we were able to do under two weeks. So maybe just to end off with maybe a teaser.

You know, Morgan, you worked through this, like, really exciting example of, like, walking through this, COBOL, a code base into Java. But one of the things is, like, in the future, what if we didn't have to modernize code itself? So maybe do you wanna speak a little bit more about this in terms of, like, the future state and how we can start thinking about this, once we already have these initiatives underway?

Morgan Lunt
54:02 - 55:26

Yeah. So in in my work with enterprise customers, I spend a lot of time thinking about the SDLC kind of as a whole and how it's changing.

It used to be that the bottleneck was writing code, and that's not really a bottleneck anymore. Like, I I consider code writing to be largely solved.

And the places where folks are seeing bottlenecks today kind of have to do with, validation. They have to do with, code review.

They have to do with integration testing, and we've got some stuff in the pipe that's hopefully gonna help people with that a little bit. And my vision is once we smooth that path a little bit and we make the process not only of writing code, but verifying, validating, and deploying code, quick and, you know, on rails and not necessarily needing to involve a whole ton of human effort through that process, we can start to do some really cool things, like keeping your code always up to date.

If there's a new Python version bump, why doesn't your package just go auto update itself and and make sure the regression test speed passes and go deploy? I'm imagining a world not too long from now when modernization isn't really a conversation anymore. It's not a thing that people have to do.

Once you get to a current state, you can guarantee that you're always, on the frontier by making incremental changes every week, every month to your code, and staying ahead of any large modernization project that would be the result of code kind of sitting static for some time and then beginning to atrophy.

Harsh Patel
55:26 - 55:41

Yeah. That is certainly resonating.

Whenever we speak with customers, many of them have a lot of these, you know, quarterly cycles where they have to go through these exercises. So to be able to do that, will completely sort of change how we think about, software altogether.

Belinda Neal
55:41 - 55:58

That was so great, Morgan and Harsh. Thank you so much.

And just one last point to make is on the docs tab of the webinar, you'll find all the links included here. So, please do check them out.

And don't forget to send us feedback, after the webinar. And we look forward to your questions and further comments.

So, thank you, everyone, for joining us today, and we hope to see you again soon.

Harsh Patel
55:58 - 55:59

Thank you.

Morgan Lunt
55:59 - 56:00

Thanks, everyone.