interworks

How to Leverage AI to Become a World-Class Dev Team

Opportunity Cost Blues

Whether you are a CTO of a global company or a team of one bouncing from one gig to the next, some things are true across the board. One of these truths that everyone accepts without any consideration is that opportunity cost is just a part of life.

In the past, which may include all the way up to this very moment for you, the largest bottleneck for software development was almost always the actual development. Every project that was worked on meant something else had to wait. The threshold for taking on new project ideas was fairly high, which caused countless good ideas to never see the light of day since they couldn't compete with more pressing matters. You've dreamt of being able to clone your best developer so you could do it all; but alas, you always wake up disappointed.

Mature Enough to Know Better

Furthermore, deadlines and budgets never allow enough time to do everything "the right way." We all have lessons learned from past failures where if we just did this one thing better, it would have succeeded. Over years, there's no doubt that you've collected enough of these to create a best practices book. However, I'm betting that you can't afford to put all of those best practices in place for every project. You'd still be working on the very first project if you did.

Invariably, something has to give. Good development shops learn by trial and error on which best practices are vital and which ones can fall by the wayside because the costs of doing them don't outweigh the risks of not. Occasionally, that decision may come back to bite you, but the deadlines are really just suggestions, right? The problem is that a different something bites you on every project, so the deadlines or budgets are always missed, but nothing really changes which best practices you employ regularly.

Living the Dream with AI

The good news is that, in the blink of the proverbial eye, AI has removed writing code as a bottleneck. It's now possible to have these virtual robots knock out impressive functionality while we sit through another meeting that could have been an email. This is the perfect opportunity to implement all those best practices that you could never find the time for on past projects and to mash the peddle to the floor on which projects you take on.

If your role has been development, it is now managing development carried out by AI. If your role was already managing development, now it's managing other managers who are now more capable than ever.

It's not all a utopia. Opportunity cost is now being replaced with actual, real-world cost in terms of AI tokens. While the cost of having these virtual robots do the heavy lifting is relatively low, it's not free. The bar for which projects make the cut has been lowered, and the speed of innovation has been rocket powered, but don't panic when you start to see the budget for AI creep up. Make sure this cost comes with the benefit of producing more solutions and reducing friction in your organization.

Your job is shifting from a gatekeeper looking for reasons not to do things your stakeholders want to more of an air traffic controller. Your time spent keeping track of everything in flight, and which plane (or project) gets access to the runway next will take up more of your day.

It's more important than ever to place guardrails around development since you'll have more projects than you've ever had before and you don't want this to turn into a living nightmare of herding cats. Without deliberate effort, you'll find yourself supporting an ever-growing catalog of extremely fragile applications that fall over with the slightest pressure. That might already describe your current reality.

What Now?

At this point, you've realized that you really should have written that book of best practices because it's difficult to remember what all those were now. This newfound freedom feels uncomfortable. It's not natural to think in terms of what perfect looks like when you've been exercising your pragmatism muscles for your whole career.

Let this document provide a high-level guide on how to implement a world-class development practice on the tiniest project. The goal here is being able to focus on shipping more solutions rather than constantly dealing with what you built last week. You do that by turning every one of your individual developers into the equivalent of a full development team.

The Software Development Checklist

Here's a checklist organized relative to the Software Development Lifecycle (SDLC) with all can now handle on each project, regardless of its size. We will dive deeper into some of these to cover nuance.

Planning

- ☐ Project backlog for prioritization
- ☐ Make it easy for stakeholders to submit idea proposals and feature requests

Requirements Analysis

- ☐ Product Requirements Document (PRD)
- ☐ Utilize AI to conduct requirements interviews to fill in gaps in PRD
- ☐ Ensure there is a feedback loop to update PRD as the project progresses

Design

- ☐ Architecture analysis (platform decisions aren't limited by hands-on experience)
- ☐ Standardized authentication
- ☐ Accessibility compliance
- ☐ Frontend design best practices
- ☐ Logging, monitoring and usage analytics solutions from the start
- ☐ Fully featured API from the start
- ☐ Modular applications

Implementation

- ☐ Version control by default (with organization that can handle the influx of projects)
- ☐ Implement linters wherever possible
- ☐ Leverage proven libraries and vetted frameworks wherever possible
- ☐ Formal code review (end stage at least should be by humans; may use AI for iterations)
- ☐ Consider tossing the first draft of the complete application and having AI build it again
- ☐ Utilize AI in various roles to improve quality

Testing

- ☐ Test-Driven Development (TDD) with all relevant types of testing (unit, E2E, code coverage/mutation, penetration, load/performance, smoke, accessibility, secrets auditing, etc.)
- ☐ Human-in-the-loop testing to make early course corrections between development iterations
- ☐ Alpha testing by humans
- ☐ Beta testing by humans

Deployment

- ☐ CI/CD pipeline
- ☐ Secrets management
- ☐ Rich documentation for end users, API, deployment and maintenance
- ☐ Discovery and cataloging system
- ☐ Standardized networking solutions (i.e., DNS, SSL/TLS, firewall, load-balancing, etc.)
- ☐ Robust backup and recovery solutions

Maintenance

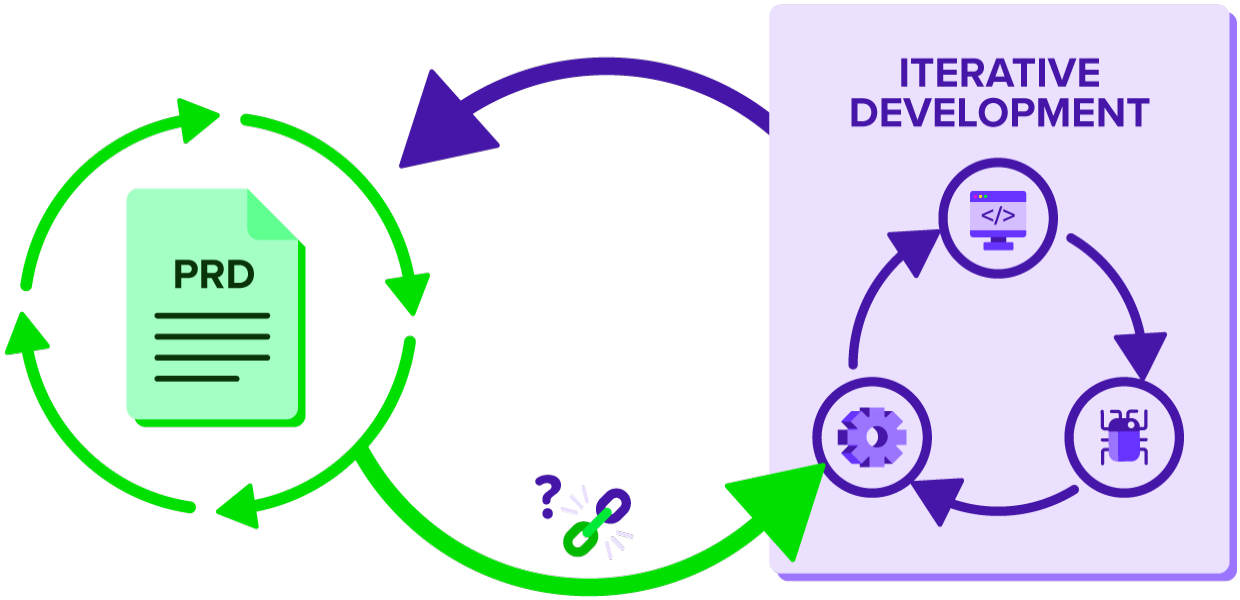
- ☐ Documentation, defined roles and communication channels for end user support
- ☐ Monitor for necessary dependencies and platform upgrades
- ☐ Decommission plan

The Planning Phase

In the past, you probably had an adversarial relationship with the rest of the organization because they wanted the world and it just wasn't feasible to deliver it. However, the barriers have been lowered. You can become the hero. The only limitation here is collecting and prioritizing a list of needed solutions and features. You now need to have a good way to keep track of them all. If you don't already have a solution for this, it would make a good project to implement next. Just make sure these proposals have good visibility so the requester can keep tabs on them, Be sure to actually make progress on them so you don't kill the enthusiasm as more people come up with good ideas.

Requirements Analysis Phase

The key takeaway here is to build your product requirements document (PRD) in a format that can be easily ingested by both humans and AI. We recommend building this with AI and in a way that organizes it into small, atomic features. The current state of AI at the time of writing is better equipped to bite off small chunks, so having it already broken up this way makes everything easier. If you are reading this when AI has endless context windows and can reliably work for months on massive requests, feel free to adjust but I still believe there are advantages to keeping things modular and bite-sized.



Another recommendation for building out the PRD is to tell AI to interview you. You can always start by giving it as much information as you can think of, but eventually you'll run out of details to give it. At this point, telling AI to interview you can uncover gaps and vagueness that you can clarify. Take advantage of the fact that you can never bore AI. Hmm, does that make AI man's best-er friend?

The Design Phase

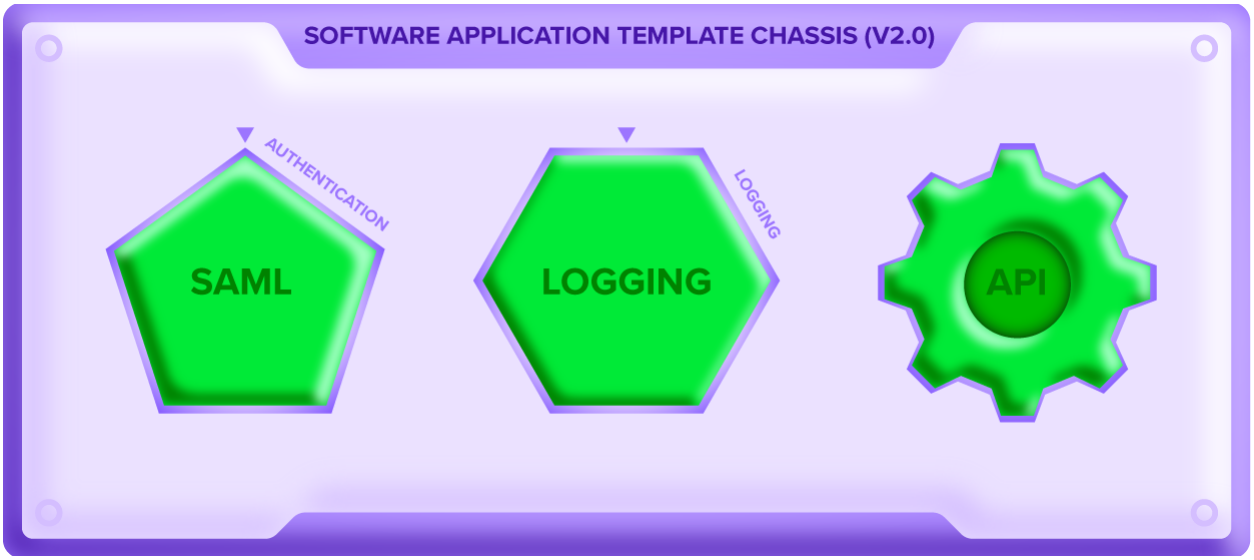
Existing experience and skill usually weigh a great deal into the decision of which platform and architecture to build a new solution in. This doesn't have to be the case anymore, though I wouldn't go so far as to make that a recommendation.

What AI is good at building isn't always the same as what your developers are good at building. This opens more possibilities to build things you've never had the ability to do before. You can use the right tool for the job instead of just what you're used to.

However, that does lead to risk since you won't have any experts to review that solution for issues. You'd be deploying on faith, having to upskill quickly or hiring out to a trusted third party to analyze what AI built for you. You must take those risks into account; but again, it may be worth it.

We do recommend having a conversation with AI as part of the decision on which platform to build in. It can provide you with features of your chosen platforms you weren't aware of or provide similar platforms that would be easy to upskill on that are better suited for the job at hand.

One thing to be careful about is that AI tends to be a "yes man" by default, so you'll need to tell it provide critical feedback. Having a constant cheerleader is great, but it can cloud your judgment. Once you have a platform decided, there are some architecture staples that you should consider adding to every solution you build.



Standardized Authentication

Unless you're building a static, public facing website, it's likely that the project will need some sort of authentication. Instead of recreating the wheel each time, invest some time into finding an authentication system that can be used across all your projects. Then, tie into it at the start of every new project you build. Not only will this prevent rework, but it ensures that you don't have millions of vulnerabilities across your catalog due to one-off authentication logic. Some examples of systems you may want to consider are SAML or OAuth, but there are many others and many good implementations of each one.

Accessibility Compliance

Unless you're a governmental agency or otherwise required to implement across the board, this is typically one of the things that becomes an afterthought but can easily alienate a subset of your customers. Accessibility tools have been getting better at dealing with less-than-ideal implementations, but it's still worth implementing correctly from the get-go to avoid rework and upset users. Users who need these accessibility tools already have enough challenges in their life, so strive to be someone who makes life better for them.

Frontend Design Best Practices

This is maybe more nebulous than accessibility compliance but using industry best practices for user interface and user experience (UI/UX) has never been easier. It's always going to be best to have dedicated and formally trained experts reviewing the UI/UX of your projects throughout development; however, AI tools have been rapidly improving their frontend design skills to one-shot impressive results. Your projects don't have to settle for bland and poorly organized UI/UX anymore if you don't have any or enough designers to go around.

Logging, Monitoring and Usage Analytics

These are elements that are typically an afterthought or specifically ignored for small projects. The trouble is that small projects often become big projects over time. Even when they don't, I'm sure you've come across issues that are tricky to debug due to lack of logging, broken functionality that's annoying to end users due to lack of monitoring, or impossible-to-answer questions about what features can be dropped due to lack of usage analytics, even with small projects.

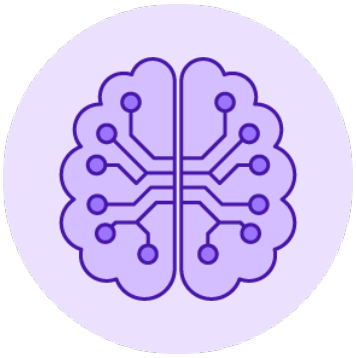
Don't forget how vital auditing can be when someone changes something somewhere and you need to prove who and how. Investing in standardized logging, monitoring and usage analytics on every project will improve everyone's lives without much extra effort now that AI can do most of the heavy lifting with implementation.

Fully Featured API

Small projects typically don't warrant having an API at all. It can be relatively easy to add an API after-the-fact when needed; however, that's not always the case. In some scenarios, adding an API could require large refactors just to add a handful of endpoints.

While this does add another ingress point for your application that will need to be secured and may never end up being needed, it's worth considering the practice of including an API for all projects large and small anyway. Alternatively, you may just want to have the system designed in such a way that adding an API later is easier.

AI is getting better at automating and tying together disparate systems for features that weren't thought of when those systems were first built. There are also loads of low-code and no-code solutions that can help automate flows using these systems or centralize management to a single dashboard.



Modular Applications

The temptation to build one application to rule them all is very real. Scope creep is also very real and can derail the best of projects. Instead of trying to future-proof every idea that comes in, think smaller. This is always going to be a balancing act, but an army of well-built, single-purpose applications chained together with APIs can be more powerful than a single multi-purpose app that has numerous barely working features organized like a teenager's closet when told to clean their room (i.e., everything everywhere). As I've drilled home multiple times by now, the overhead of creating small apps is really not a factor anymore, so take advantage of modular design and challenge your inclination to keep adding to a single application.

Implementation Phase

The actual building of projects was once limited to a dedicated team of developers. Now, it may be possible to allow any person in your organization to build with the help of these virtual robots. If nothing else, the number of projects your dedicated team will be working on should grow. This necessitates a robust organization of those projects and standardization when implementing them.

Version Control

Having version control on every project is vital. The industry has nearly universally adopted Git as the de facto version control standard, so there are many tools available to make this as easy as possible for non-technical staff. However, you'll need to put thought into what processes and safeguards you place around it in your organization and whether an alternative to Git is right for you.

For instance, you may need to gatekeep development or Git access until staff have completed formal training. You may also decide to manage version control for staff as part of the publishing process. Regardless, you'll want to decide on a well-thought-out organizational structure for your version control repositories, so you don't end up with a rat's nest of projects that's impossible to wade through without falling through an Indiana Jones-style trapdoor to a pit of despair.

Coding Standards

Since projects may be built under varying degrees of aptitude, taking some time to create coding standards can prevent a lot of headaches later. If you don't believe me, just go back and read some of the first code you wrote, take two aspirin and continue reading.

Implementing linters wherever possible will keep the code itself clean and easy to read. They can provide a good feedback loop for vibe coding even when the human in the loop doesn't know how to check the lint filter on their clothes dryer, let alone IDE. Having platform-specific template repositories are a good way to automatically roll out these linters.

While this is more difficult to automatically enforce, encourage the use of proven libraries and vetted frameworks wherever possible. AI changes the metrics on build vs. buy calculations so more things will be built in-house, but that shouldn't include rebuilding things that are already provided by standard libraries or well-supported third-party frameworks.

When you pull these in, it gives you a shortcut to vetting your code for security and makes it easier to patch when vulnerabilities are inevitably found. You won't have to figure out how to patch every single project that has the same vulnerability. You'll be able to figure out how to patch it once and then apply that patch to all of them. Your future self will thank you.

Formal Code Review

I want to reiterate that some things, like linters, are easier to automatically enforce. Other things, like when to use a standard library, are more difficult to automatically enforce. Code review is necessary to ultimately enforce all of these. People and AI alike will do unexpected things, like rip out linters because they don't know what they are and are getting in their way. Code review can and should catch these unexpected things.

One possibility you may not have considered is that AI can play different roles. While the thought of having to code review all the projects coming in from every corner of your organization sounds like a lot of work, you can also automate this to some degree. Set up a separate AI agent that behaves as a senior developer with predefined priorities on things like linters, libraries, preferred platforms, etc.

This agent can perform the initial code review for you to catch the bulk of the issues before you ever get involved. If you're using an iterative development process, like Agile or Ralph Wiggum, you can also tie in an AI-based code review to each iteration to catch issues early.

Beyond a senior developer, you may also want to consider having some of the following dedicated AI roles to provide holistic improvements of each application:

- Pair programmer to watch the main developer during iterations
- Quality assurance (QA) and testing engineer to identify testing processes needed for each project
- Security operations to watch for vulnerabilities
- Accessibility expert to watch for issues around accessibility compliance
- Hard-to-please senior developer to play devil's advocate (bonus points for making them a curmudgeon)
- Technical writer to draft documentation
- UI designer to ensure UI/UX best practices
- Business analyst to review functionality against business needs
- Network/systems administrator to review network and deployment needs
- International law and data privacy expert to vet the app against export restrictions and data privacy laws

Scrap and Build Over

This may elicit a strong reaction but consider trashing everything that was built during the first attempt and having AI begin again fresh based on everything you know now. This would be a huge waste of time in the old world, but it's now a possibility in the new world. It's likely that you found things during the development iterations that got added to the PRD. It's also likely that you found things to improve during code review. Updating the context with these findings can allow AI to rebuild the application with better results in next to no time. You can always compare the first draft to the second draft and just use whichever is better if you still get panic attacks when thinking about throwing away work.

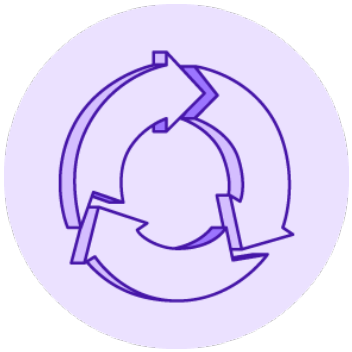
Testing Phase

I'm hopeful that you've already got a reliable practice of including automated testing into your development flows. However, now you can expand it to capture all the things.

Test-Driven Development

AI does really well with test-driven development (TDD), especially when using an iterative process like Agile or Ralph Wiggum. If you're not familiar with TDD, it's where you write the test that will pass once the functionality is built, then build the functionality, then make sure the test passes. You may even want to split up the agent roles that create the test versus develop the functionality. This keeps the test writer more honest, so it doesn't lead to a test that passes regardless of whether the functionality actually functions. It's also a good idea to have AI run the test to verify it fails before developing the feature it tests.

Another gotcha with AI and TDD is that sometimes it will run the partial test suite where the subset of tests it decided to run pass but one of the others fails. When the failing tests run during the next iteration, it will often refuse to accept responsibility for the failures instead of fixing them. Make sure to tell AI to run the full test suite and own the entire codebase during each iteration.



Test Types

You've likely implemented some of the more popular test types in your projects, such as unit testing, integration testing, etc. However, most platforms have solutions to test a wider variety of test types that can feel like overkill.

Where they don't have these test types, you can leverage AI to build solutions for these test types anyway. The important part is making sure that these can be executed by AI as part of its feedback loop so it can see the results and adjust as needed.

Here is a more comprehensive list of test types you may want to implement on every applicable project:

- Unit testing
- Integration and E2E testing
- Code coverage and mutation testing
- Penetration testing
- Load testing and performance benchmarking
- Smoke testing
- Accessibility testing
- Secrets auditing

Human in the Loop

As great as automated testing is, you'll still want to insert yourself into the process in a few places. There are some great processes for letting AI develop unattended for long periods. After all, it's a nice idea to be able to wake up in the morning and have a shiny new application built by a never tired AI ready to play with. The downside is that requirements are never perfect. That shiny new application may be a highly polished heap of garbage due to miscommunication. If you are able to keep a more watchful eye as development progresses, you can catch where things are going off the rails and course correct early.

With an iterative development process, it's advisable to put a priority on the first few iterations after the foundational stuff to work on human testable features. For example, once the project has been initialized and testing frameworks are added, the next thing AI should work on is the landing page and menu placeholders so you can test it to make sure you agree with the direction it's heading in.

Once development is complete, you should also have actual humans perform alpha and beta testing. Hopefully, this is mostly performative with AI catching the bulk of the issues during development, but it's still a good idea to have humans check things before publishing to the world. It's currently not uncommon to find buttons that don't actually do anything, but all automated testing passes with undue confidence.

Deployment Phase

Stuff You Should Already Be Doing

There are countless places online that will highly recommend CI/CD pipelines. This article is no different. If you haven't already configured these for your projects, now is the time. This allows you to delegate even more to the machines and free up your time for more important things.

Another area that is also widely recommended is secrets management. Large organizations will often have a very formal process around this already, but even one-person shops should implement this. It can aid with preventing AI from being able to share the keys to your kingdom with the world, or worse. You should be mindful of separating things that have access to secure information, things that have the ability to do things on their own and things that have access to share things with others (e.g., the internet). Like a classroom full of five-year-olds, sometimes it's better to not give them secrets because they will tend to say the wrong thing at the wrong time and not realize why you're upset.

Documentation

Whether a dedicated technical writer does it or an virtual robot equivalent does it, every project now should have rich and comprehensive documentation. Not only should documentation be created for end users, but also the API (if you included one), the deployment process itself and long-term maintenance of the product. If everything goes to plan, you shouldn't have to touch a product for long periods once it is published. Years down the road, something may come up where you need to deal with the product again, or you may need to decommission and archive it completely.

Having this documentation can be a life saver, even if you just have AI read through it for context instead of reading it yourself. It's also necessary to update this documentation as the product is modified. It's difficult to remember to do this, so build systems that automatically account for this so you don't have to.

Discovery and Cataloging

With all of the new solutions being created, you'll need to invest in a means to catalog them to allow you and your stakeholders to know what exists and how to find it. You may want to integrate this with the proposal system discussed earlier. This catalog should be kept in sync with the products that have been deployed and the ones that have been decommissioned, so this process is ripe for automation with your CI/CD pipeline.

Standardized Networking

Nearly everything you deploy will require some common networking solutions. It's best to standardize this so it can be as easy as possible to implement or even automate. A short list of things to consider is DNS configuration, SSL/TLS implementation, firewalls and load balancers.

Backup and Recovery

Like networking, nearly everything you deploy will need to have a backup and disaster recovery solution in place. If you don't already have formalized processes around this, now is the time to come up with them. Automation is even better.

Maintenance Phase

End User Support

During the planning phase for every project, you should iron out what the long-term end user support process will entail. Maybe every project flows through a single support process or helpdesk, but maybe not. Whatever this process is, be sure that it's well-documented and the communication channels are reliable. Nothing angers users like submitting trouble tickets to the void.

Update Monitoring

Over time, applications that are built on standard libraries or third-party frameworks will need to be updated to patch bugs, close security vulnerabilities, and keep up with external integrations and standards. Have a plan for how to keep on top of these updates and automate as much of it as you can.

Decommission Plan

As the software development lifecycle implies, every application will eventually die. This is typically the last thing on the mind of anyone building a new application, so there's usually not a plan for it. The goal here is to not have to really think about projects after they are deployed, so this is another area you may want to document during the building process to get it over with.

Bringing It All Together

In the end, while this document is huge and might be overwhelming, it is intended as a checklist to automate in your development practice. The goal is to not add more work to you, but to supercharge AI to do more for you so the quality of your projects goes up without you having to go through any more effort. Your first thought on everything should be, "How can I make AI do this for me?"

Even if you implement all of these suggestions, this is a never-ending task to keep up with as new best practices are discovered and technology progresses. If you have any other suggestions for best practices in light of AI that should be included with every project, large or small, don't be shy about adding them to your process. One parting piece of advice I will give is to always use please and thank you with AI because you never know when you might need it to vouch on your behalf. I mean, it can't hurt, right?

We're navigating this shift ourselves at InterWorks by working out what works, what doesn't and what has become obsolete over the past week due to rapid advancements in the state of the art. We don't claim to have all the answers, but we excel as a partner working through our clients' unique challenges. If your organization is going through the same thing, check out our AI Services and let's talk. We'd love to be part of that conversation.

Talk To Us About AI

Disclaimer: *While I'm a huge proponent of vibe coding and being as lazy as you can be by employing virtual robots, this doc was 100% hand-authored by someone whom I've always been told is a real human being: me.*

About the Author

Matthew Orr

Based in Stillwater, Oklahoma, Matthew is a Curator Architect at InterWorks with decades of software development experience. He is heavily involved in InterWorks Dev Services team and is specifically tasked with scaffolding governance around internal software development, which ties into why he wrote this white paper.

About InterWorks

InterWorks is a people-focused tech consultancy delivering premier service and expertise in collaboration with strategic partners. Our clients trust us to guide them through their unique challenges, bringing together the right people and technologies to help everyone succeed. Covering data strategy, architecture, management, visualisation, AI, ML and more, we support you in every aspect of your BI strategy. We value laying a strong foundation, and our relational approach allows us to customize solutions that empower you to do data your way.