The Four Patterns of AI native Development

Introduction

The job of a developer consists of performing many different tasks. Like any new technology, AI helps us to execute these tasks easier, faster, better, cheaper. As some tasks change, it can also introduce new tasks or make other tasks completely obsolete. This challenges us to rethink how we approach this change instead of just continuing our existing ways of working. We define this as **AI native Development**: what are the fundamental changes and where this is heading to.

The technology side of it is evolving at an unprecedented rate. From simple prompts and chat all the way to collaborating autonomous agents. The improvements are visible and the expectations high. The thought experiment is to assume that the technology continues to improve and then extrapolate how that ultimately influences the current developer role and tasks as being part of a *socio-technical system*.

Pattern overview

We've distilled a set of patterns that we will describe as shifts in focus from one task to another. It doesn't mean the original task or focus is less important but rather that AI is getting better at it , allowing us to spend more time on the new one.

1. *From producer to manager*: as AI writes more and more of the code, our focus shifts from producing the code to reviewing the code. In the extreme form AI becomes so good that we only have to deal with the exceptions and manage situations where it can't execute the task without our help.
2. *From implementation to intent*: in an ideal world, we just have to specify what we want and have the AI implement it. We feed it a set of business and technical specifications and keep chatting with it when it needs more explanation or needs to solve some conflicting requirements.
3. *From delivery to discovery:* while in the past we were able to pursue the delivery of a single idea, as implementation is becoming cheaper and faster, we can try out multiple ideas and solutions and discover which is worth pursuing.
4. *From content to knowledge:* when everyone can build their ideas reliably, the differentiating factor is the business domain knowhow and knowledge. Instead of having it buried in documents, we want to surface it at the right moment and in the right context. AI both helps us and benefits from it to produce better results.

Emerging practices

You may wonder how we got the above patterns. We did this by extrapolating various practices we see in the industry. We'll look at them through the lens of a specific pattern and notice that practices have different levels of implementation: from simple to more advanced.This has to do with the adoption level of these practices: sometimes it's not a mere technology break through but it could be a different design or an emerging need for new solutions waiting to mature.

1. *From producer to manager*:
    - From simply chatting with AI or doing code autocompletion, we learned that we need to review the code it produces as it's not always perfect or what we need.

- This continues as we're asked to review PRs for code generated by AI. It produces code more frequently and the code changes are bigger to review. And now it's not only one file anymore, but multi file changes and refactorings.
- As the cognitive load keeps increasing, we're looking for smarter ways to manage this: for example contextual diffs, step by step reviews , moldable development IDEs.
- This will be extended towards dealing with production issues as well: agents asking for our decision will require us to have situational awareness in case of emergency.
- We'll have to know what good looks like and have the AI inform us to make the right decisions. In addition we'll need to build this human oversight and interaction into our process.
- You can't become a manager overnight and it favors the more experienced people. On the other hand AI can help anyone to level up their knowledge faster allowing them to step into various parts of the process.
- It's important that humans stay in control but we want to avoid micro-managing the system. Therefore we must be able to assess the risk and the impact of choices. And that we architect for failsafe environments that reduce the risk for the business and most of all for the end user.

2. *From implementation to intent*:
   - Prompts and chat were among the first to have expressed what we want in natural language instead of having to write the code ourselves.
   - The more AI got access to our context, be it the whole code base, our documentation, the browser and the terminal, it was able to produce better results.
   - We moved into a state of continuous prompting, where even the corrections were expressed as prompts instead of us jumping and correcting the code directly. A concept sometimes referred to as vibe coding.
   - We learned we can keep track of some of the recurring conversations into a specifications documentation as simple as a markdown file and specifying an image as a crude design.
   - As we accumulate more insights, we add more requirements ranging from actual designs, technical requirements like backend architecture , coding style , performance all the way up to functional and business requirements.
   - Now we can collaborate with various stakeholders and also detect conflicting requirements. We expect Ai to be able to generate the changing requirements to changes into the code.
   - The same process can also be used to extract requirements from legacy systems and help in migration from one system to another.

3. *From delivery to discovery:*
   - Understanding what needs to be implemented has always been challenging. Agile helped us with a closer collaboration and understanding the business better.
   - Still, most of the time was spent on building and delivering the solution to production. The job was often more focused on building the thing right instead of the right thing.
   - Now with coding becoming easier and cheaper, we can delegate more of the delivery and focus more on the discussions with the business. With rapid prototyping, we can generate multiple versions and ideas in the discovery phase.
   - In addition to internal discovery , we can now also spend more time listening for signals from customers and do experiments and get feedback from production. It takes A/B testing to a new level by auto adapting to the audience.
   - Discovery does not only have to apply to the business part, also different technical options can be implemented in parallel and tested to make the right decisions.
   - And in the case of support and issues, we can have the AI generate multiple hypotheses and deliver different solutions to help end users.

4. *From content to knowledge:*

    - We used to spend a lot of time googling things or looking them up on Stack Overflow. No shortage of content available. Now with the help of AI we can bring all that information available when we need it for a specific task.
    - In addition to how to solve a problem, we also rely on documentation to deal with conventions and guidelines inside the company : from coding styles in Readme files, to dealing with incidents in runbooks to security and compliance guidelines. We have a love/hate relationship with these documents : the effort to write them and then keeping them up to date often feels like a chore. AI can help us to stay on top of it.
    - Even within a team or shared codebase whether in public or inside of a company we have a set of rules we need to follow and jargon we need to understand. It's hard to keep track of changes and constantly understand the latest. Therefore in context suggestions and learning help us navigate this space more easily.
    - Understanding an existing codebase can take a lot of time and if not documented well, we rely on almost word of mouth for getting people up to speed. Likewise , the impact of someone leaving the team will impact the delivery speed. With AI in a constant conversation with us , it can track this knowledge and capture it as one big memory including remembering past experiments and learnings.
    - Additionally incidents and support can be accumulated together with learnings from the business. We can learn at our own pace and our learnings are recorded for everyone else to use.
    - The more and more easily building things becomes, it becomes a commodity. And the differentiating factor is that specific domain knowledge, be it technical or business , is vital to stay ahead of the competition.

Part of a socio technical system

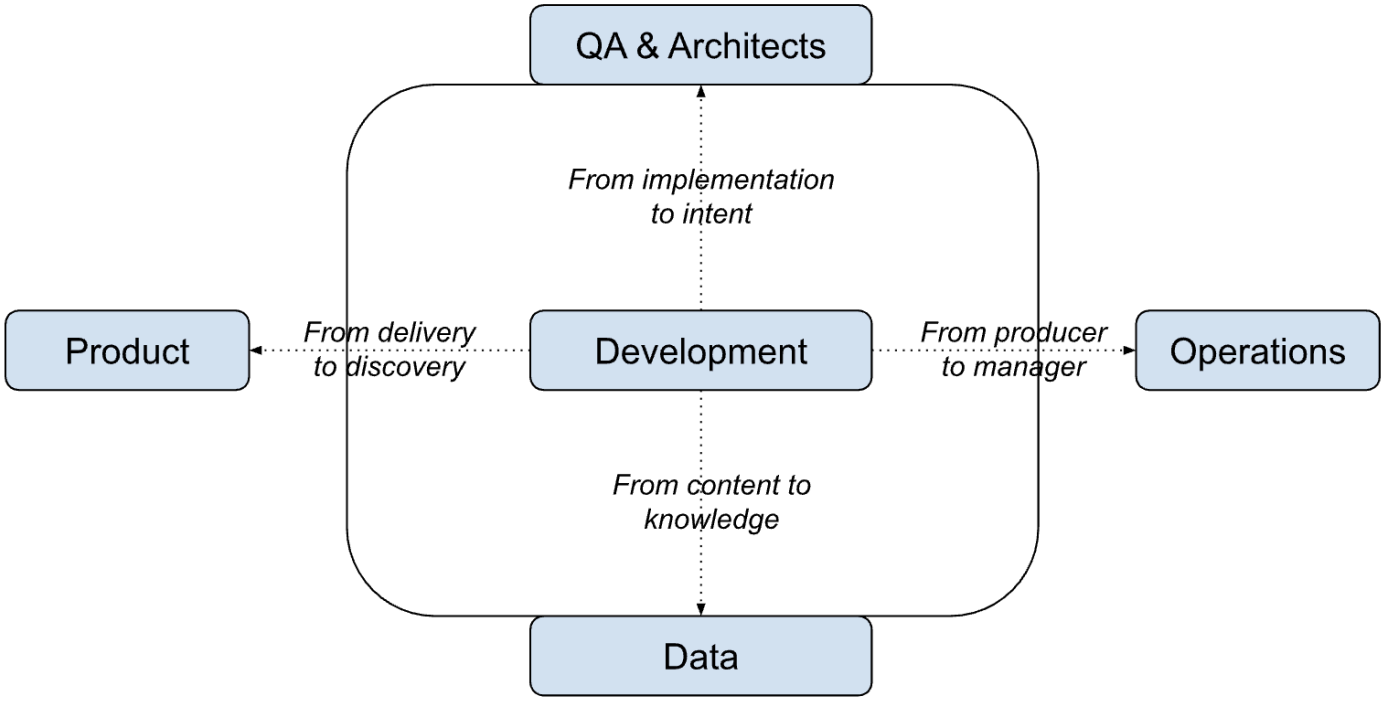
Developers are not the only ones helping to deliver value to the customer : product owners, agile coaches, engineering leads, testers and QA, data engineers, DevOps & SRE and security all play a vital role. And all roles in this social technical ecosystem are influenced by AI.

From a higher level, one can recognize the patterns as developers breaking into other roles and vice versa:

- *From producer to manager*: this is related to becoming an operator and dealing with issues when they occur. We can learn from all the DevOps observability and automation to help in this shift.
- *From implementation to intent*: typically a domain for the architects and QA people to write down and guard the specifications.
- *From delivery to discovery:* the product owner is typically focused on understanding the business , the customers and the domain.
- *From content to knowledge:* like data scientists finding signal in the noise, cleaning and extracting information of a multitude of data

This should not come as a surprise, as an organization is constant in flux and an interaction between different actors. We see shifts towards more generalists, not just full stack but also multi domain. Even before AI, good developers did not only excel at producing code but at collaborating and understanding the business domain and we expect this trend will only continue and increase.

As for training , hiring and reskilling we believe that besides still understanding and knowing what good code and delivery looks like, we'd encourage people to cross the role silo boundaries and collaborate with other stakeholders.



Conclusion

Time will tell how far and how fast this new technology lives up to its promises. We've shied away from the technical implementation details here to focus on the change in behavior. We've shown the four AI Native Dev patterns together with the various practices that accompany them. In addition to the shifts in tasks, roles and responsibilities between the different organisational groups, we'll also have to see where in the organization AI will come into play. We hope the patterns give you guidance on where to learn and explore new tooling and help make sense into this ever changing world. We'd love to hear your thoughts on this and also we'd welcome challenging us with practices you see that don't fit these patterns. The only way is to learn together and in the open and improve our story.