The YAGNI Principle Stops Devs From Getting Ahead of Themselves

Why build something now when you can put it off forever?

Tatum Hunter | Associate Editor

Tatum Hunter is a former Built In associate editor covering software engineering, design and UX, and software sales. She currently covers personal technology for The Washington Post.

December 14, 2021

Updated: February 15, 2022

When Marie Dingess talks about the benefits of agile practices in Capital One's software departments, she's not just paying lip service: She started there in the early 2000s, before the company used agile at all.

She remembers an effort almost two decades ago to build a brand new content management system. About two months before the anticipated deployment, company leadership shifted, priorities changed and the project was abandoned.

"After months of hard work, it just went by the wayside," she told Built In in 2020. "It might have been the right business decision at the time, but the team's morale was

Developers should start building capabilities only when those capabilities are actually needed, rather than trying to predict future needs.

Dingess, who is now director of agile coaching at Capital One, learned firsthand what can happen when teams, for reasons good or bad, violate the extreme programming principle known as YAGNI: "You aren't gonna need it."

Tammy Xu contributed reporting to this story.

What Is YAGNI?

Behind the "You aren't gonna need it" mentality is a central warning: Sometimes planning too far in advance only serves to make projects harder.

There should always be a balance between trial and error and planning in software development. Developers who swing too far into the second category run the risk of overengineering their projects. Developers think they're saving time by creating a feature now for customer requests they expect are coming down the line, but sometimes their expectations are wrong. That's why agile stresses the importance of tight and continuous feedback loops with customers — otherwise, you can wind up doing a whole lot of work on something that will never be needed.

But the real danger of overengineering is how it may impact current deliverables. If future expectations are baked into different parts of the codebase and then those expectations are wrong, developers have to go back and remove the changes. That delay can cause development teams to miss deadlines on current deliverables and leave a messy codebase behind.

particular feature or functionality, build it, and build it quickly. Otherwise, wait.

Understanding YAGNI's definition is easy; implementing it is harder. Developers who have worked with disorganized product managers may worry that future features won't have the necessary back-end support. Teams that evaluate devs based on productivity will also find programmers eager to work ahead on product roadmaps.

> **"You should know when your team is doing it, because you'll hear things like, 'While we have the hood open.'"**

For example, a team Dingess once worked with was busy updating a database when some developers realized an upcoming feature on the roadmap called for a new field. So, they decided to create the field while they were already in the database. This, Dingess said, is when their YAGNI alarm bells should have started ringing.

"You should know when your team is doing it, because you'll hear things like, 'While we have the hood open,' or, 'While we're here, it makes sense to go ahead and do this,'" she said.

Sure enough, when the developers reached the roadmap feature that called for the new field, the request had changed, and they had to go back and do that work again.

READ THIS NEXT

Software Estimate Debates Aren't Really About Estimates

Image: Shutterstock

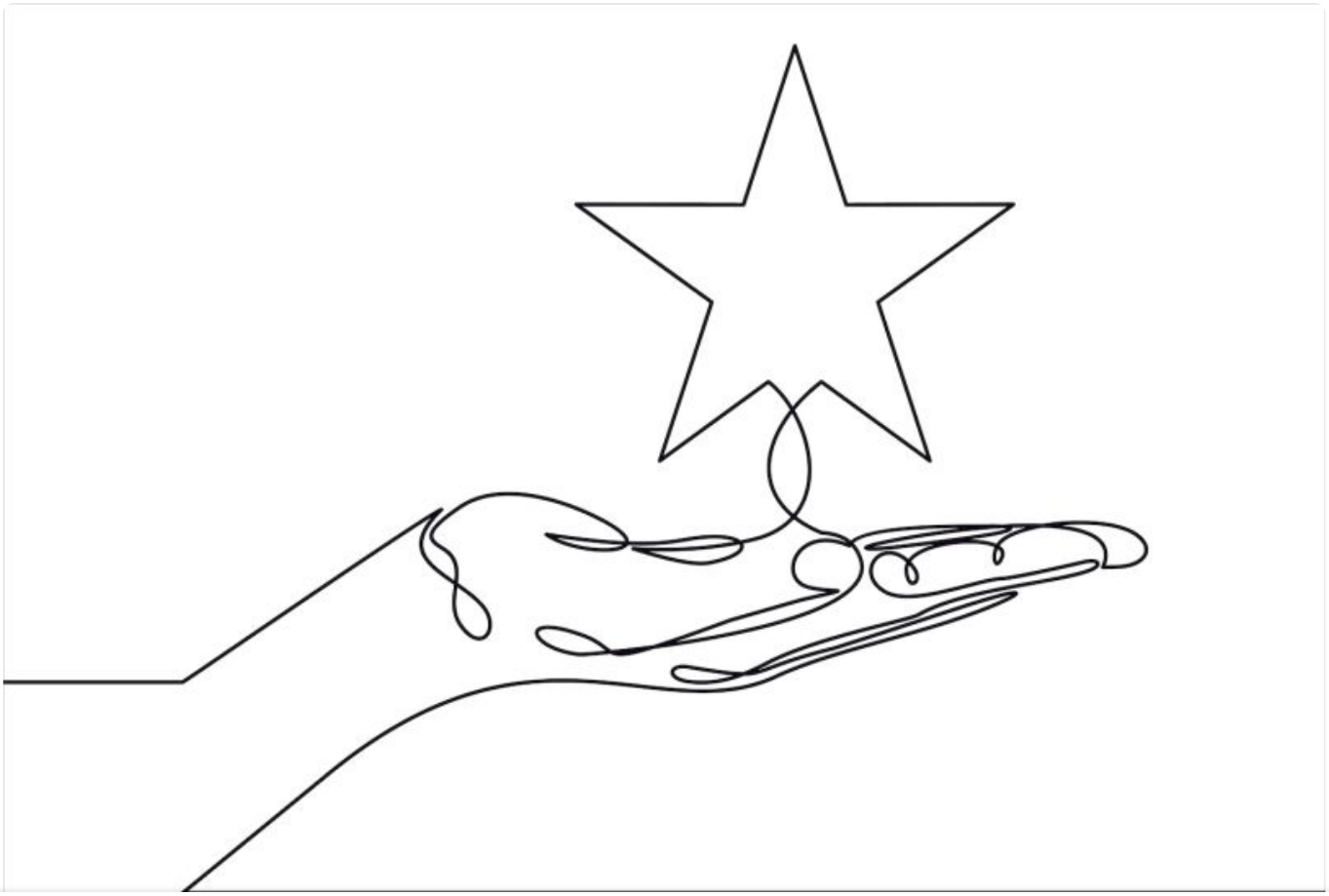
It's Hard to Say No

Any conversation about YAGNI should also be a conversation about work in progress, and how many responsibilities are on an engineer's plate at one time, Dingess said.

Engineers (rightly) view themselves as problem solvers, and it can be hard for them to say no to requests. But prioritizing work based on what's needed at the moment — and knocking out one thing at a time — is a better approach than trying to multitask.

Dingess has teams go through a simple exercise to simulate what happens when developers don't use YAGNI to pare down their work in progress. First, she asks participants to write down the names of everyone on the team, one letter at a time. Then, she asks them to write the names out all at once and see which method takes longer.

The point of the game is clear within seconds: Of course, it's more reasonable to work on one thing at a time. But in the moment, it's still tough to explain to a stakeholder that engineers can't dive into their request for weeks or months.



should.

Small-scale experiments, in the form of minimum viable products, are a good way to respond to feature requests without violating the YAGNI principle, Dingess said.

For instance, if a webform needs a simple input field, developers could connect the field to a rudimentary API instead of setting up client-side validation. Then, if the webform proves necessary, they can increase the fidelity from sprint to sprint, adding more validation and front-end prettiness.

Working prototypes and incremental integrations are a better approach than building something fully deployable and hoping that specs stay the same and integrations go smoothly. If the feature doesn't achieve the desired business outcome, it can be scrapped without much drama.

> "We had to say 'no' or 'not yet' to about 30 features."

But what about when product, marketing or customer success professionals start requesting full-blown projects with little consideration for incremental design?

"Form your conversation this way," Dingess advised: "'I need to test this out. And to do that, I'm just going to focus on this part first. I'm going to come back and share with you if it was successful. Then, we can discuss the next thing we need to do to move forward.'"

After Capital One's transition to agile, Dingess found herself on a team tasked with building a new digital product. Determined not to repeat the content management system debacle, product and technology leaders came together to determine which features were absolutely necessary to prove that the product provided real value. Product managers mapped some user stories, and engineers identified the simplest architecture required to address those user needs.

Instead, the team pivoted to build out the product based on that feedback.

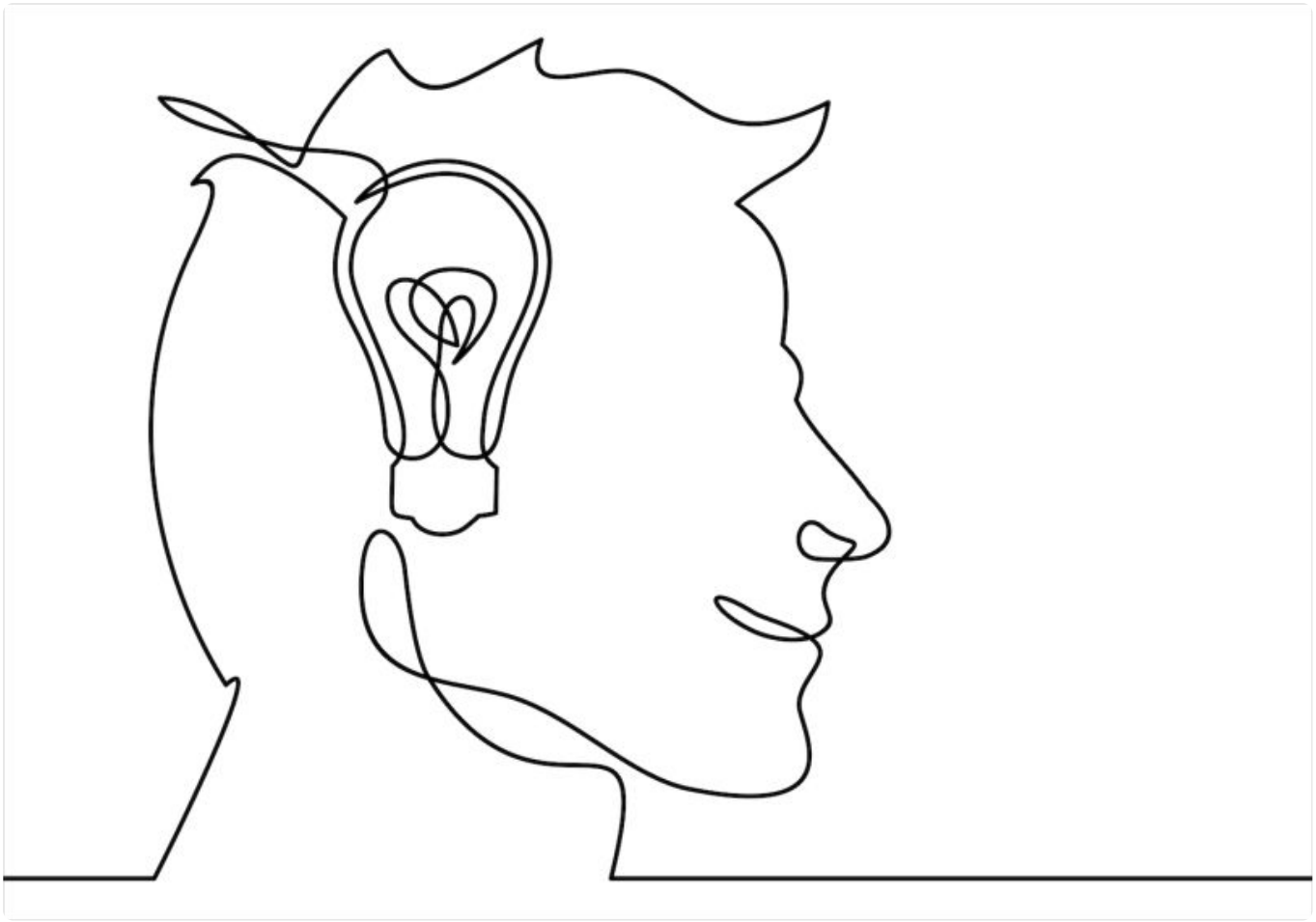


Image: Shutterstock

YAGNI 201: Beyond the Build

When YAGNI popped up as part of extreme programming, it was referring to software development: You aren't gonna need that capability you're building for the future.

But YAGNI-style thinking has proved useful in plenty of other contexts: Why are we

asking.

Duarte, who wrote the book on the #NoEstimates push, sees #NoEstimates as a natural extension of extreme programming.

"If I want to be tongue in cheek, I'll tell people that #NoEstimates is agile turned all the way up to 11, just like XP was," he said. "XP is the unsung hero of the agile movement. And I hope that with #NoEstimates, we're bringing that mentality to how we make decisions, how we manage backlogs, how we make investments and how we manage organizations."

Teams that successfully work without estimates are regularly deploying products to market in extremely short development cycles, Duarte said. Rather than planning and executing big projects, they pay close attention to the impact of small changes to the product — and adapt constantly.

"They spend more time deciding what not to do than trying to discuss the architecture of something they don't need but will build anyway," he added.

If that sounds familiar, that's because it's YAGNI.

> **"YAGNI was a technical practice when it was discussed in the XP world. Now, we're talking about it not just from a technical perspective, but even from a business perspective."**

Duarte refers to the implementation of #NoEstimates as "The Clinton Process," after an interview he conducted with game development expert Clinton Keith. Keith took a project slated for 18 months and knocked it out in two weeks through some artful applications of YAGNI-style thinking.

Here's how it works: When a new product idea gets tossed out, instead of "We aren't

those solutions for value.

“You don’t expect that doing the work is what’s needed, you expect that delivering value is what’s needed,” Duarte said. “Even before the software is written, test for value with paper prototypes, user interviews, whatever.”

The last step is to give yourself a “ridiculously short” time box. Duarte encourages teams to follow each user story for half a day. If they can demonstrate value with a bare-bones solution, they can then keep building on it as long as they’re adding value with each build. If the story takes longer than a half day to address, there’s something wrong, and the team needs to return to the we-don’t-have-time-for-that phase.

If half a day sounds absurd, consider this: In his #NoEsimates workshops, Duarte has participants take a sample project that would normally take six months and deliver value after 30 minutes.

And scenarios like that aren’t limited to workshops — they happen in real life too.

Once, Duarte worked with a company that needed to deliver regulatory reports to its clients for them to sign and send back. The company wanted an entirely digital

Turns out, it would.

That conversation probably took a few minutes. The other five months, 29 days, 23 hours and 40 minutes? You aren't gonna need them.

"YAGNI was a technical practice when it was discussed in the XP world," Duarte said. "Now, we're talking about it not just from a technical perspective, but even from a business perspective."

READ THIS NEXT

What Do Software Consultants Actually Do?

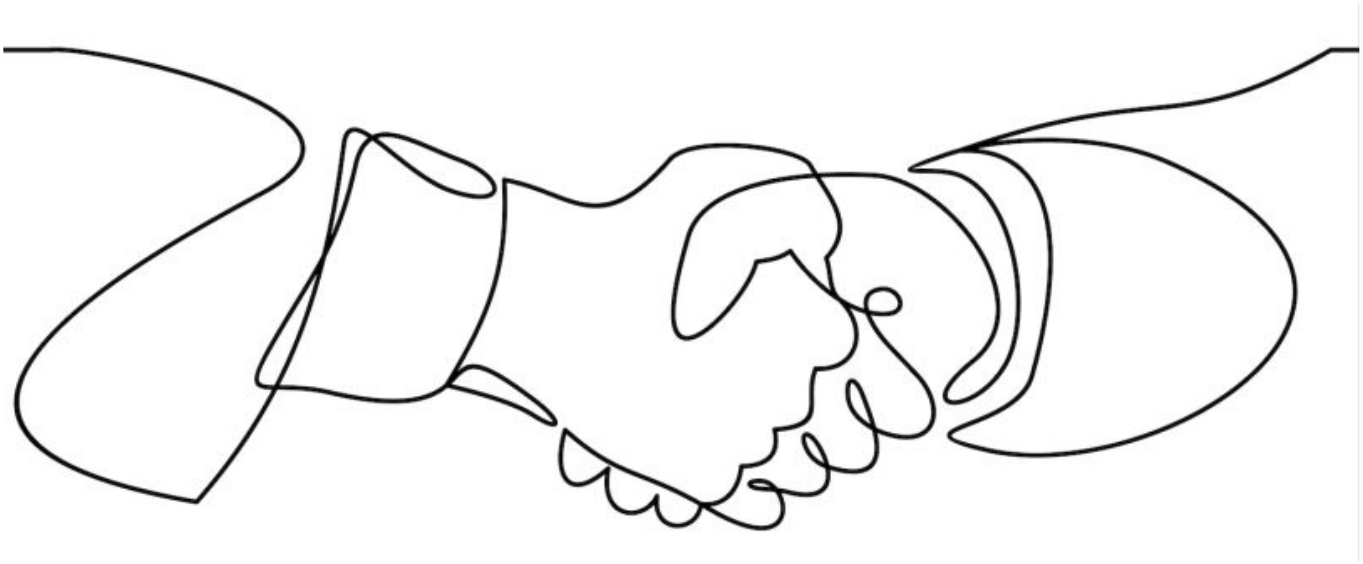


Image: Shutterstock

You Aren't Gonna Need Traditional Contracts

Another business application of YAGNI is agile consultant Jacopo Romei's extreme

materials contracts. For someone doing predictable, replicable work — like a construction expert — that makes sense. The final product either looks like the blueprint, or it doesn't.

When hiring knowledge workers, though, those models force companies to act against their own interests, Romei reasoned. A contracted software developer could spin up a solution that doesn't add any value and still earn a fixed rate. Similarly, a time-and-materials agreement would incentivize companies to guess at the best solution in an attempt to control for costs, rather than letting a developer experiment and test for value.

So, Romei decided to take the principles of extreme programming and apply them to contract negotiations. Long contract terms and fixed goals? You aren't gonna need them.

Instead, Romei started using short-term, value-based contracts with clients. He agrees to work on a problem, with no predetermined solution, for a short amount of time. If his solution creates business value in the form of revenue bumps or cost savings, he makes a percentage of that value added. If he doesn't, he either makes a very low fixed rate or walks away with nothing.

"If you're afraid that your work will not be valuable to someone, then what's the point in wasting time and resources? To protect yourself more than you need?" Romei asked. "Basically, I don't want to be signing contracts in an attempt to not lose. I want to collaborate for a win."

> **"If I'm an idiot, I want to give them the right to discover that I'm an idiot very early."**

This approach, Romei said, solves a lot of issues at once.

have a website," Romei said.

Second, it prevents contractors from over-engineering their solutions in order to work longer and make more money. An extreme contractor takes a company's problem and tries to address it as quickly as possible. If that solution adds value, the contract might get renewed for another short period. (This method also protects the contractor's time and income, as an unsuccessful partnership won't suck up months of work.)

"This is not a wedding. A business collaboration is hanging out for some time and maybe renewing that relationship," Romei said. "Instead, contracts are always shaped as if we're agreeing to never disagree, which is impossible. I want the option to get out of the collaboration for free. I want the option to disagree with you."

"At the same time," he continued, "I want clients to have the option to collaborate with me, and not the obligation. If I'm an idiot, I want to give them the right to discover that I'm an idiot very early."

Third, value-based contracts mean contractors have more skin in the game. Rather than delivering the minimum acceptable solution that justifies their rates, they're incentivized to deliver the solution with the maximum value. In other words, there's no upper bound to performance, as Romei calls it.

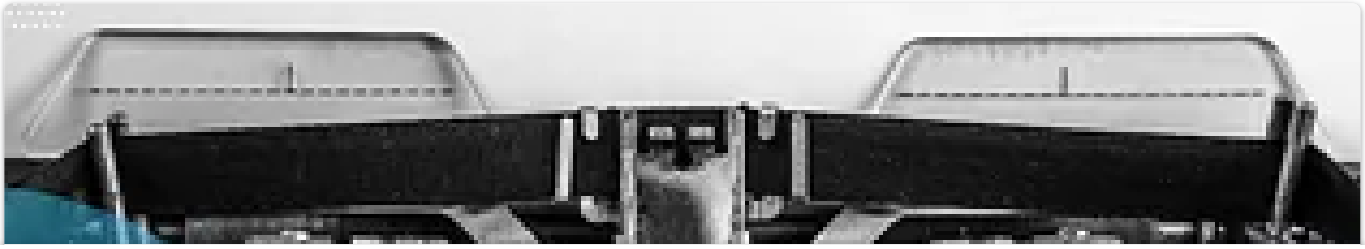
"It's YAGNI, in that as long as we shape collaborations that aren't killing us if they go wrong, we aren't gonna need any predictability," he said. "Let's expose ourselves to the upsides of collaboration. Let's expose ourselves to overshooting our goals, in a good way."

Despite the prolonged popularity of agile and extreme programming, plenty of people still view both as a cut-and-paste set of engineering practices, rather than a set of values with broader implications for the workplace. Hopefully, Romei said, new

wear leather and know all the lyrics.

RECENT PRODUCT MANAGEMENT ARTICLES

21 Health Apps to Know



Culture Is Key. And at Innovid, There's No Shortage of It.

Product Management | Software Engineering Perspectives

FOR EMPLOYERS

Our Story

Careers

Our Staff Writers

Content Descriptions

Company News

Get Involved

Recruit With Built In

Become an Expert Contributor

Send Us a News Tip

Resources

Customer Support

Share Feedback

Report a Bug

Browse Jobs

Tech Hubs

Built In Austin

Built In Boston

Built In San Francisco

Built In Seattle

See All Tech Hubs