Software as Craft

I've been doing ABC ("always be coding") lately. I also admired an Icelandic sweater at the CTO Club yesterday. And it got me thinking about the connections between the two.

Mathematics and knitting have a long history together (spoiler alert: I accidentally opened my birthday presents from my sister early: thank you so much AJ for Making Maths with Needlework and Crocheting Adventures with Hyperbolic Planes), and early hackers like Babbage reused equipment (e.g. punched cards) from the textile industries for their code. But I'm not sure whether creating an app has been described in terms of knitting a sweater before. So here goes.

We have the basic knitting stitches (stockinette, garter, ribs etc) - these to me are like the languages in computer science (Ruby, Python etc). Each of them is composed of smaller things (individual stitches/if statements, themselves created from lengths of wool/characters) which can themselves be more or less complex, but the basic idea of a foundation is there.

Going up, we have the framework that we're building in - for a garment, that's the shape that we're building it in (you thought a jumper was just a front, back, a neck and two arms? Oh boy...), for a language, that's a framework like Django, Rails or Kohana. Both of these have a set of modern conventions (e.g. ORM) that come out of earlier and for-a-while-forgotten idioms.

But even if you have a framework and a stitch in knitting, that's not the end of the story. Knowing Python and understanding Django doesn't make you a good app developer. And knowing cable stitch and raglan shaping doesn't make you a great Aran sweater (and no, you don't want to know what a three-needle bindoff is, honest!). What makes a great Aran sweater is years of looking at other people's Aran designs, years of actually getting down and knitting different designs to really understanding the history of why some subpatterns keep recurring. Basically, if you want to be a great knitter, then knit. Lots (it helps to have small relatives - the mistakes take less time to create, and generally said relatives don't care so much when your dinosaurs look like strangely-shaped ponies). Design your own sweaters and learn from them why the classic designs are shaped the way they are. Seek out experts, examine their work in detail (in knitting, many of those experts are long dead - in computing, not so much at the moment), listen - really listen - to their experience, and ask "why" they did things a particular way. And just plain keep on building stuff.

And so it goes with code. If you want to be a great coder, then code. Read other people's code. Make mistakes. Commit to something sizable of your own design and learn the hard way. Understand not just the stitches and the shape, but also the reasons for the patterns that you see (and which you rarely see tutorials for: language and framework tutorials are easy to come by; tutorials on how to best structure frameworks, not so much...).

Hmm. Agile. That, perhaps, is more of a patchwork quilt... you can start small and add pieces, swap out pieces that don't work, but at some point early in the process, changing out the top-level

design (unifying patterns and colour scheme) becomes really really painful.

I could head off on a tangent about regional differences in coding style now, but I won't.