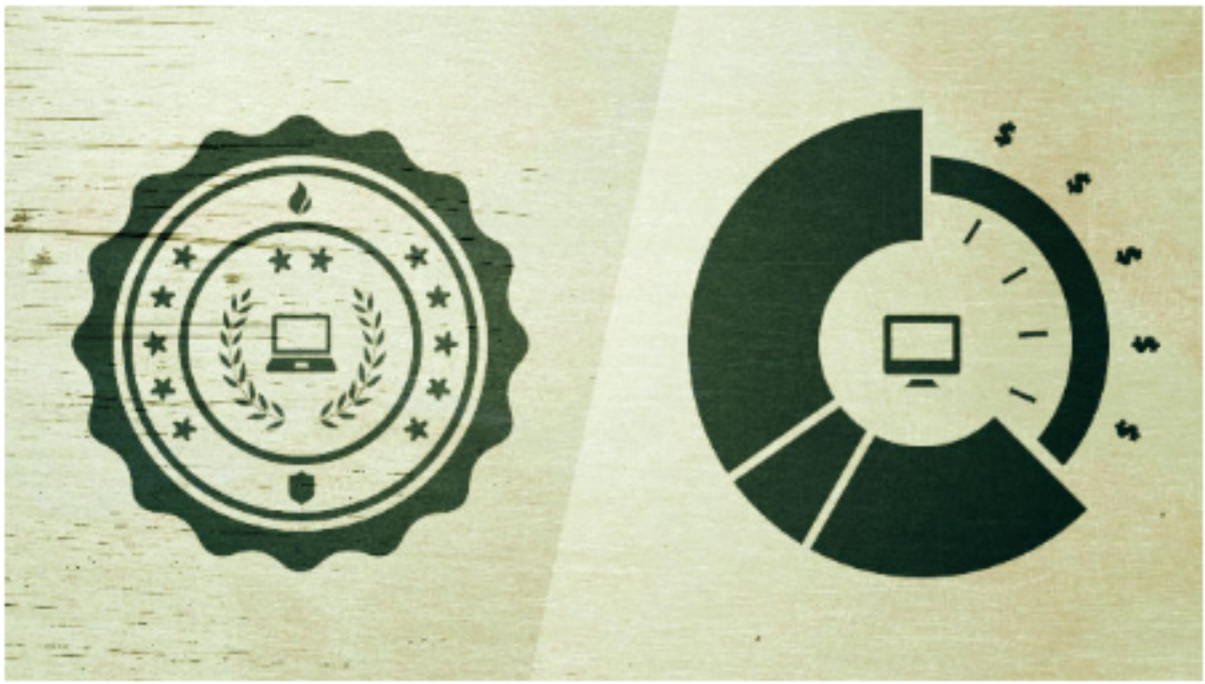
SIGN IN

TECHNOLOGY LAB —

What's the difference between college-level and corporate programming?

You'll have to make a leap to get to "paid professional" status.

STACK EXCHANGE - 12/1/2013, 10:00 PM



Stack Exchange

This Q&A is part of a weekly series of posts highlighting common questions encountered by technophiles and answered by users at Stack Exchange, a free, community-powered network of 100+ Q&A sites.

rdasxy asks:

When they graduate and get their first job, a lot of students feel like they don't really know how to program even though they may have been good programmers in college.

What are some of the differences between programming in an academic setting and programming in the 'real world'?

See the original question here.

Programming alone does not a programmer make

Michael answers (54 votes):

In a traditional undergraduate computer science program you learn *just* programming. But the real world doesn't want people who are *just* programmers. The real world wants real software engineers. I know many job descriptions don't seem to express this distinction, which only confuses the matter. In the real world you need to be able to:

- Gather and analyze requirements when they aren't directly given to you.
- Design and analyze architecture with near endless possibilities.
- Create test plans and act on them to evaluate and improve the quality of a system.
- Work collaboratively on a team of people with different backgrounds and experience levels.
- Estimate and plan work even if you don't know exactly what to build.
- Communicate effectively with stakeholders who have different needs that don't necessarily align.
- Negotiate schedule, budget, quality, and features without disappointing stakeholders.

Oh yeah, and you also have to be able to write code too, though that takes, on average, only 40 to 60 percent of a software engineer's time.

So, it's not that freshly minted computer science undergrads don't know how to program (many are in fact, very good programmers). It's that many of them don't know how to do anything else.

Code in a vacuum

back2dos answers (22 votes):

At university, your teacher gives you:

- A well defined, isolated problem, the solution of which can be provided within a short and well-defined time span (and it will be discarded afterward).
- A well-defined set of tools that you were introduced to prior to assignment
- A well-defined measure for the quality of your solution, with which you can easily determine whether your solution is good enough or not.

In the "Real World":

- The problem is blurry, complex and embedded in context. It's a set of contradictory requirements that change over time and your solution must be flexible and robust enough for you to react to those changes in an acceptable time.
- The tools must be picked by you. Maybe there's already something usable in your team's 10-year-old codebase, maybe there's some open source project, or maybe a commercial library library will have it. Or, maybe you will have to write it on your own.
- To determine whether the current iteration of your software is an improvement (because you're almost never actually done with a software project), you need to do regression testing and usability testing, the latter of which usually means that the blurry, complex, contradictory, context-embedded requirements shift once again.

Conclusion

Programming in school and programming in the real world are so inherently different to the point where there's actually very little overlap. CS will prepare you for "real world" software development like athletics training would prepare an army for battle.

School is simple

Mike Dunleavy answers (6 votes):

Good answers. Let me just add, academic programming tends to be almost toy-like in scale. This is good for teaching. As a teacher, you are trying to convey ideas most efficiently. The downside is realistic programming is so qualitatively different, it's hard to bridge the gap.

One area of difference is in performance analysis. I've written many posts trying to point this out. Performance analysis is only superficially about algorithms and measuring. To do it really effectively, you have to approach it as a process of debugging.

Another area of difference is maintainability. This encompasses everything from style to domain-specific language design. You can't do it effectively unless you actually know what you're trying to minimize.

These things are not taught, and they make an enormous difference in productivity.

Real world code

dimitris mistriotis answers (5 votes):

Update: As if someone was reading my mind: "Graduate expectations versus reality"

My take, two other factors:

Problem size: In academia, I mostly had to develop software "from scratch," which meant that most of the time, the largest program I had encountered was the largest one I wrote. This de-emphasises the necessary capability to handle and cope with complexity that emerges from different pieces of software interacting together. If I was aware of the effort needed to comprehend with complexity, I might have chosen not to be in the industry at all.

Reading vs. Writing: Another side effect of problem size is that often, in the "real world" we are exposed to work that has been written by others, either for maintenance purposes (I did no maintenance in academia anywhere), extension, or simply division of labor. Therefore reading code becomes many times more important than writing it.

A proposal for improved programming education: Academia should expose us more to real-world situations without regressing to vocational training. Doctors have to face a corpse at some point to see if they are "made for it" (I've heard stories of people dropping the course after this experience). If I had seen in my early twenties a 20K LOC project comprised of different programming styles, which I had to understand in one day and amend a bug in three, I might have considered other career options —though probably not.

Related: "What should I expect from my first programming job?"

Find more answers or leave your own at the original post. See more Q&As like this at Programmers, a question and answer site for professional programmers interested in conceptual questions about software

development. If you've got your own programming problem that requires a solution, log in to Programmers and ask a question (it's free).

READER COMMENTS

SHARE THIS STORY

← PREVIOUS STORY

NEXT STORY →

Related Stories

Sponsored Stories

Powered by

Today on Ars

RSS FEEDS
VIEW MOBILE SITE
VISIT ARS TECHNICA UK
ABOUT US
CONTACT US
STAFF
ADVERTISE WITH US
REPRINTS