LISTENING TO EARLY CAREER SOFTWARE DEVELOPERS *

Michelle Craig
University of Toronto
mcraig@cs.toronto.edu

Phill Conrad, Dylan Lynch, Natasha Lee and Laura Anthony
University of California, Santa Barbara
pconrad@cs.ucsb.edu

ABSTRACT

Previous work finds that recent college graduates entering the software development industry encounter difficulties early in their careers, due to significant differences between their coding experiences in academia vs. what is expected of them on the job. To explore the gap between academic and industry software development, we conducted interviews with twenty early career software developers with four-year degrees in CS. We present an analysis of these interviews, including excerpts in the developers' own words, organized around six themes. We conclude with thoughts on how to bridge this gap so that our students may be better prepared when we launch them into their careers.

1 INTRODUCTION

Students that graduate from four-year degree programs in Computer Science pursue a variety of career paths. Some pursue graduate degrees in Computer Science, or professional degrees in law, medicine, or business. The vast majority, however, directly enter the workforce-most, at least initially in career paths involving software development.

There is no shortage of opinions from faculty about what an appropriate undergraduate curriculum in Computer Science should look like, and the extent to which it should, or should not, be driven by perceived or real software industry needs. In this paper, we listen to a different set of opinions-those of graduates from four-year degree programs in Computer Science with between one and five years of industry experience. We hear about their experiences of stepping into the world of software development, and whether or not they felt well-served by their university preparation. We analyze

interviews with twenty recent graduates, identifying common themes about the gap between the preparation that a CS degree provides, and the skills and knowledge needed to be successful in a software development career. We also surveyed previous work in this area.

Our literature survey found that this area has been studied for at least a decade with a variety of methodologies, resulting each time in similar recommendations for curricular improvements. But while small curricular innovations have been discussed in experience reports during this time, on the whole, very little has changed – a gap persists. Our paper makes three contributions: (1) a framework for characterizing the gap organized around six specific themes that emerged from our interviews (see Table 2), (2) first person accounts of this gap by developers, in their own words, and (3) some thoughts on why the gap has persisted, and how the computing education community might respond.

2 RELATED WORK

Table 1 lists five previous studies related to early career developers. Begel & Simon performed a "fly on the wall" study of eight recent college graduates during their first six months at Microsoft. The first of two resulting papers focused on abilities, deficiencies and misconceptions of early career software developers [2], while the second applied ideas from a research field known as "newcomer socialization" [1]. Sudol & Jaspan examined differences between the beliefs of students and industry professionals about Software Engineering [11], focusing on particular misconceptions. Hewner & Guzdial interviewed and surveyed professionals at a particular computing gaming industry employer to determine what skills were valued in game developers [4]. Radermacher & Walia addressed the gap between student preparation and industry expectations in a 2013 literature survey [8]. Exter investigated the skills gap in a 2014 report of a study of educational software developers [3]-like our work, it includes an analysis of interview transcripts.

Our work further explores the skills gap for students from university CS programs who move to software development positions. Ten years have passed since Begel & Simon conducted their observations and called for universities to change their curricula to incorporate the use of legacy code projects-a long time in an industry that changes as rapidly as software development. Some individual curricular interventions have been reported including the use of Open Source Software [6] and studio-based learning [7]. We wanted to know: does the gap persist, or have things changed? Like Exter, who interviewed developers of educational software, and Hewner & Guzdial, who interviewed the game-development community, we listened to developers first hand in semi-structured interviews. Unlike these two previous studies, our interviews are not limited to developers from a particular industry subdomain. In contrast to Begel & Simon, our study relies on self-reporting via interviews rather than direct observation, is not focused on employees of a single company, and is limited to participants that have completed four-year college degrees.

Table 1: Related Studies

S1	2008	Begel & Simon [1,2]	"Fly-on-the-wall" observation, 8 early career developers at Microsoft, post-study interviews
S2	2010	Sudol & Jaspan [11]	surveys of 45 industry professionals and 115 students
S3	2010	Hewner & Guzdial [4]	surveys of 7 leaders and 100 professionals at a game development company
S4	2013	Radermacher & Walia [8]	systematic literature survey of 30 papers (from pool of 14,968 candidates from 1995-2011.)
S5	2014	Exter [3]	interviews with 9 developers from educational software industry, online survey (N=74)
	2018	*(this paper)*	20 interviews with early career developers from 15 companies across four U.S. States

3 METHODOLOGY

We used personal contact networks, word of mouth and developer meetup groups (i.e. convenience sampling) to recruit developers who consented to participate in an interview of no longer than one hour. The first author conducted 20 interviews (nine face-to-face, the others via teleconference). The interviewer had a previous relationship with two of the 20 subjects but did not know the others. All the interviews began with the following grand-tour question [9]:

I'm interested in hearing about the software development that you do on your job and in particular how your undergraduate education prepared you (or didn't prepare you) for what you do. Can you think back to your first year as a developer after graduating and tell me your story?

The subject was encouraged to tell as much of their story as possible without interruption, with the interviewer indicating ongoing interest but trying not to influence the conversation. Once the subject ran out of story, the interviewer continued with follow-up questions in the form of a semi-structured interview, allowing the researcher to seek clarifications and ask about a few specific topics of interest if those were not already covered. [10] The interviewer used this list of possible follow up questions, adjusting the wording as appropriate to subjects' previous responses:

- Which of your courses or university activities do you think were most useful in preparing you for your job?

- In what areas were you under-prepared? well-prepared?
- What percentage of your work time is spent on software development activities? (coding, designing, testing, etc.)
- What fraction of this is on legacy code (vs. greenfield)
- Did you have legacy code exposure during your undergrad education from some course or activity?
- Knowing what you know now, do you have any suggestions for an undergraduate CS program that would prepare students to be software developers?

Except for one interview where the interviewer took extensive handwritten notes, audio for interviews was recorded and transcribed. All participants in the study were developers with between one and five years of work experience after graduation. The subjects spanned 15 companies, four U.S. States and came from four different universities. The companies represented ranged from tiny startups with a handful of employees to small businesses with between 50 and 100 developers to "household name" giants with more than 50,000 employees.

All five authors met, read each transcript aloud, phrase-by-phrase. For each phrase, we discussed and assigned one or more codes. We did not start with pre-defined codes-in each case, we discussed whether to assign an existing code, or create a new one. All coding was done by consensus, thus inter-rater reliability is not an applicable metric.

Table 2: Software Development in Academia vs. Industry

where	academia	industry	S1	S2	S3	S4	S5	this paper
what	well-defined, fixed-scope	vague, open-ended, evolving scope		✓		✓		✓
when	short time spans (days, weeks)	long time spans (months, years)		✓			✓	✓
who	individuals, pairs, small groups	larger teams	✓		✓	✓	✓	✓
why	to learn something	to address a user need				✓		✓
how	ad-hoc tools and practices	professional tools and formal practices		✓		✓	✓	✓
how big	small programs	large complex systems	✓				✓	✓

Columns S1–S5 refer to the studies from Table 1 providing evidence of the importance of this axis.

4 ANALYSIS AND DISCUSSION

Our analysis of the interview transcripts revealed six specific dimensions of the gap between the experience of writing code in industry vs. what students do in their university programming assignments. We re-examined the studies listed in Table 1 in light of these six axes. For each, a checkmark appears in the columns corresponding to studies presenting evidence that this axis is important aspect of the gap. The remainder of this section presents a discussion of these six themes as found in both the previous work, and in our subjects' own words. Bold numbers P01 through P20 indicate the participant numbers and quotations are italicized.

4.1 What: Differences in Scope

In school, most assignments are well-defined and narrowly scoped. Instructors or TAs write specifications which, in order to make grading manageable, have well-defined deliverables. In contrast, our new developer participants experienced poorly specified and open-ended requirements.

P10 *Sometimes it [the program he is writing at work] is not as well thought out. Like in school, it's very well thought out. The tests are always written for you and you make the tests pass. Or the project is really well defined. Like the scope of the project is so well defined. But here, like half the time we don't know what problem we're solving.*

Some participants pointed out that project courses in school were closer to their current work in terms of scope.

P20 *Mostly the project courses were the best preparing me ... [because they] give you a very limited time schedule. So you have to ... learn how to scope things, how to adapt to different situations. Also, because you're working with a bunch of other students who also have their own priorities and some of them have different work ethics.*

Many interviewees talked about projects done outside of school that helped them learn and practice concepts not well-covered in their school experience such as fault tolerance, scale, deployment and project-management infrastructure. Even when schoolwork includes projects, participants noted that the requirements specifications in industry have a lot more give-and-take than the fixed requirements of a school assignment:

P19 *Some other things I didn't learn in school were ... arguing about the design, not arguing, but debating about the best possible solution for the product requirements as well as the engineering requirements because you know PMs can dream up all sorts of great things for their product, but in the end someone's got to implement it, and sort of that give and take is a process.*

P14 suggested that to better prepare students, instructors should assign programming tasks and then intentionally change the specifications after students have partially finished the coding.

4.2 When: Short vs. Long Time Spans

In school, students typically write code for an assignment, then move on to the next one, seldom (if ever) revisiting old code. The time span is days or weeks-at most, a few months. In contrast, industry code lifespan is more typically years-developers inherit legacy code from their predecessors and must maintain, interface with or extend this code.

Inspired partially by Begel & Simon's observations in this area, we asked each developer to estimate the fraction of time they spent on legacy code vs. greenfield development. The answers ranged from no time on legacy code to 95% of their time. Those in small startups indicated less legacy code work. A few developers indicated that when first hired they spent a larger fraction of their time on legacy code but had recently started a project with more new code development. Some noted that even new code is effectively becomes legacy code in short order.

P16 *Unless you're starting your own startup from scratch or you're so experienced that they're going to give you a brand new project, chances are you're going to be inheriting some legacy code*

P10 *I would say a lot of it feels greenfield, but in reality almost, I mean anything that you make greenfield within a month is now legacy. I wrote something a month ago, and I just looked at it last night because I have to kind of do it again but in a slightly different way, yay, and now I'm looking at it and I'm like 'Why did I, (pause) I don't even know what I'm doing here. What is going on?'*

P14 *I'm doing almost all of the work by myself, so it's all green (pause) or you could say it's legacy after a day because you still have to maintain it.*

The fact that essentially all industry code quickly becomes legacy code that has to be maintained highlights the importance of good design and good style (a theme we return to in Section 4.6). Unless curricula are specifically designed to force these issues, students may not appreciate this until they arrive in industry. Most developers we interviewed had not worked with legacy code while at school:

P06 *The biggest difference between school and real life is that everything I did in college was greenfield and almost nothing since.*

When we specifically asked about school experience with legacy code, six participants (from two different universities) mentioned their operating systems course. Some of these said that the OS course was hard because of the large pre-existing codebase but others indicated that it was still not as difficult as the challenges faced at work. One pointed out that the starting codebase in the OS course was well-written code.

P06 *It was different [than the legacy code at work] because the other parts worked pretty well and you didn't really have to understand all of it or have to wonder how to fit in your stuff.*

P14 and P15 (from the same school), independently mentioned a specific course they took that incorporated maintenance of legacy applications:

P16 *[This] class was really cool in the sense that it kind of showed you here's how you pick up a project that's already existing and then make modifications to it to meet some new spec while, at the same time, you're collaborating with your team to keep the code as understandable to everyone as possible.*

P14 *That was the class I probably learned the most in. He immediately like, dove us into Github, and um, we were forced to use version control and forced to work with legacy applications and that kind of like, gave me the best experience for the industry.*

Later in the interview when P14 revealed that he spent all his working time doing greenfield development, the interviewer probed further about the usefulness of this classroom experience.

Interviewer: So you mentioned having an experience with legacy code in your undergrad. Do you think that was valuable even though you're not doing it now as a developer?

P14 *I still kind of do it, because I'm like "what was I actually trying to do here?" (pause) so with the legacy stuff in [professor X's] class, you almost (pause) we spent a lot of, somewhat of the time trying to figure out where in the code we should make a change. So that was a good experience to have done.*

Most of the developers indicated that their jobs now, but even more so in their first few months, involved spending a lot of time working with legacy codebases. Almost all of them indicated that this was a struggle and while a couple had experienced legacy applications in their coursework, others expressed that there was an opportunity for university programs to better prepare students for this. In contrast, P7 indicated that while he of course didn't specifically learn the proprietary software from his new company, he felt that school had prepared him well for dealing with large amount of legacy code claiming, "I had the tools."

Two of the most strongly-held misconceptions from Sudol & Jaspan are directly related to legacy code vs. greenfield development, Exter calls for programs to "bring existing large-scale real-world applications and infrastructures into the curriculum" adding to the earlier calls from Begel & Simon and Sudol & Jaspan to incorporate more than greenfield development into university assignments.

4.3 Who: Individual vs. Large Team

All of the related work mentioned the importance of teamwork for software developers. For example, in Hewner & Guzdial's survey, "ability to work with others and check your ego at the door" was the most-essential qualification for employment. Yet Sudol & Jaspan still find persistent student misconceptions about the value of teamwork.

Our participants also report it as an area where they were insufficiently prepared. When the codebase you are working with was written by numerous previous developers, finding the right people at your company with the expertise you need-and interacting with them-was often a significant challenge (in contrast to a college course context where the professor and/or TAs are typically the obvious ones to ask.)

P20 *You need to be able to figure out which people on the team to ask questions to-and then ask them the right sorts of questions to get the information you need, to work with the code that you're dealing with.*

P19 *A lot of understanding it [a large codebase] is also like finding who to talk to about which part.*

Another theme stressed repeatedly was being unprepared for the degree of teamwork required in their jobs.

P10 *the one thing I felt like ... I wasn't prepared for was ... I didn't have much project experience on like, working as a team.*

P17 *that's what I do on a day to day basis; I work in a team with three other devs and uh, we do have to divide and conquer the work.*

P07 *the classes where you worked in groups in Computer Science was (sic) very useful to me ... My advice would be focus on group project ... I think that's very critical to people coming out of college and working in a big business like this. You need to know how to work with other people.*

P11 *They [my instructors] didn't want us to work in pairs. It was all individual work whereas in actual-going into the field it was all about asking questions, getting help from everyone else.*

Participants stressed time management in a team context:

P09 *The first thing I noticed when I got into the workforce is like-"wow this is like we're all working on a huge group project together" - so I definitely think just as far as like personal management and time management skills and working with people in group projects -it definitely even applies today in my current role as a software engineer*

4.4 Why: Learning vs. User Needs

Radermacher & Walia reported a study[5] from 1999 that found that, "educators did not place as much value on teaching students how to prepare for and conduct interviews with users as did industry practitioners." It appears that this is still the case 18 years later. Four students emphasized the importance of customer considerations in industry work, with three students noting that a majority of their development time was spent with customers.

P10 *The engineering that I do a lot of the times is talking with product and deciding what the problem is ... me trying to understand what exactly the problem they're seeing is*

P12 *most of the time [my work is] altering existing code to meet the needs and requirements of the customers*

P06 *Customers wanted things certain ways and making them happy was more important than other considerations*

4.5 How: Ad-Hoc vs. Professional

Our developers told us that Agile software development practices, which are very relevant to industry work, were rarely or never mentioned in undergraduate courses, and if so were poorly covered.

P03 *as far as what's extremely relevant for the job ... source control, testing, agile principles, weren't really mentioned*

P08 *I didn't really understand agile*

They felt that not using version control makes the projects completed at school unnecessarily harder, echoing a concern from Exter's survey, where "working with change control software" was the least-well covered topic with over 75% of the 53 respondents who had taken computing courses indicating that it was not covered at all. This may be attributed to the Exter's participant population, however Radermacher & Walia and Begel & Simon both also cite tool use (particularly configuration management and version control) as deficiencies. Our students reported this as well. P15 mentioned how the use of Github in particular enhances a developer's resume:

P11 *I didn't know about github or git or anything related to that and that would have been helpful to learn ... every company has something managing their code online and usually it's github*

P15 *Teach github at a first level ... you cannot go to a company these days and not know github ... [Github is] basically your online portfolio*

P10 *whenever I used code in school we would literally email it. Like when we were working on a project and on a team, I remember emailing code. I remember sending someone a dropbox file. And I remember like copying and pasting it wrong and getting really frustrated and thinking I was really dumb because I couldn't figure out the program, and in reality I just missed a semicolon somewhere. And it was because I copy and pasted it wrong*

Some subjects also expressed the need for skills in implementing and maintaining production level programs.

P08 *[During school] I never had worried about "productionizing" services like being very fault tolerant. Never worried about monitoring a service, or alerting, or manually monitoring jobs*

Testing is also important in industry work and developers argued that it wasn't sufficiently covered in school.

P16 *I feel like I would have had a much easier time doing projects on complicated issues if I had known how to prevent stupid errors via knowing things like testing, code quality and you know, preventing giant strings of code that are untestable.*

4.6 How Big: Small vs. Large Codebases

Participants said that they spent much more time adding features to large existing codebases than writing stand-alone programs. When asked about how school could have better prepared them, subjects indicated that they wanted more emphasis on reading code, using libraries, good design and refactoring. One developer explains that having good code style has become more important now that he creates code so rapidly he no longer remembers all the details, while another developer explains how much her code style has improved since starting work in industry.

P01 *I wish there was more of an emphasis on integrating different libraries.*

P06 *[Programs should] focus more on reading code and working in other codebases.*

P04 *[Students should be required to] have the framework of some system built out and then build new features for an existing system, so you have to understand the system before you can go and design something that builds on top of it.*

P08 *Learning how to refactor period was one of the biggest things. I feel like in a lot of academic settings, you write your main function and you sort of overflow one file with a lot of things to do something that you're tasked with and its not. There's no focus on how you are going to build an architecture how the system will function with singular responsibilities.*

P10 *it [writing code] has become so routine for me that I don't remember what half of it is and if you don't have some sort of rules about what's going on then you're like, (pause) I don't know. So it's like 'what does this function do?'*

P19 *[When I was at school] I wrote awful code. Lines everywhere, you know, just hard to read. At [company name], we're very strict about following [company name] style guides. ... They're very strict. We follow them to the letter. Learning how to structure my code like that, actually I think it's definitely, I mean it helps within a lot of the codebase*

having everything follow a certain standard, but also I think the rules in the style guide generally do make the code a lot easier to read.

5 INTERNSHIPS

Although we didn't set out to explore internships, many of our participants indicated that during their college education, they had done one or more internships in industry. Of those that did, most indicated that it was the closest experience to their current situation. In some dimensions, particularly "how" (professional tools) and "who" (larger teams), internships are closely aligned with the industry software experience. But in other dimensions, our developers told us that an internship is often more like a university experience: many interns are sheltered from the frustrations of legacy code and the demands of real customers by being given projects that are more "stand-alone" (rather than tightly integrated with legacy code) and low-risk (vs. mission critical). The timelines are often short-the length of one internship-and because the "user" is often the supervising developer, the specifications can be more narrowly scoped than typical work. These observations are in agreement with those of Sudol & Jaspan who find that students who complete internships have fewer misconceptions overall, but their likelihood of having a misconception about working alone or about the accuracy of client descriptions of requirements does not correlate with having completed an internship. We conclude that internships are useful in addressing the gap, but not the full answer.

6 THREATS TO VALIDITY

Our participants' comments on their university training reflects their own programs and schools. Even within the limited scope of universities offering four-year CS programs, 20 interviews with students from four schools may not be enough. While our coverage of company size was well spread, the web development industry may be over-represented, vs. the software industry as a whole.

We attempted to reduce to threat of subjects telling us "what they expected us to want to hear" by having an unknown person conduct the interviews where possible. However, the interviewer did know two of the subjects and the initial contact for many of the others was from someone they knew from school. So while minimized, this threat was not eliminated.

7 CONCLUSIONS AND FUTURE WORK

Our interviews reveal that many new software developers feel ill-prepared in multiple areas including tool use, communications, teamwork and working on large, long-lasting, open-scoped, complex software systems. We claim that this is due to the fundamental differences between the programming activities typically done in university coursework and the coding done in professional software development environments. In spite of nearly two decades since the gap in industry/academic coding experiences was identified and nearly ten years since researchers made recommendations for curricular changes to address the gap, it is still quite wide.

Academics often note that the purpose of university education is not vocational training (e.g. "we are not a trade school"). This point of view is not without merit. We are not suggesting a wholesale course correction, but rather a healthy middle ground that balances a rigorous training in Computer Science fundamentals with helping students feel well prepared for their careers.

We note three main obstacles to progress. The first is **awareness** and acknowledgement of the problem. We addressed this by giving voice to our former students concerns directly, in their own words. The fact that their concerns echo ones raised in previous studies suggests they are not anecdotal or isolated. The second is **familiarity**: we hypothesize that our oft-overworked faculty are at times ill-equipped to teach students about professional practices and large, complex software systems because they themselves have never crossed the gap. To address this, we suggest that faculty may be well served by taking some time to have authentic encounters with industrial practices, perhaps through a "summer internship in industry" similar to those our students experience. One of the faculty authors of this paper experienced just such an internship and found it eye-opening, and helpful in transforming that instructor's courses.

The third is likely the most difficult to surmount: namely the **difficulty** of creating authentic experiences of various aspects of the industry software development context in academic settings-especially doing so at scale. There have been successful efforts with small numbers of students (e.g. [7]), but scaling these efforts to the enrollments of large programs is daunting: imagine, for example, the challenges of managing a class of 200 students in pairs, with each pair working on a different project in teams-perhaps using Agile processes such as Scrum or Kanban-with some of these projects having dependencies on one another. How can one provision these projects in a way that addresses concerns of consistent evaluation and grading, equitable learning opportunities, and academic integrity, to name just three concerns-not just in a senior capstone project setting, but throughout the curriculum?

These are difficult questions. Yet, somehow, software managers in industry do manage large groups of developers, working on disparate projects. Perhaps the next group of our former students to whom we should listen are those that have gone on to careers managing such organizations, to see what we can learn about how we might restructure our courses to bridge the gap.

REFERENCES

[1] Begel A., Simon B., Novice software developers, all over again, *Proceedings of the Fourth International Workshop on Computing Education Research*, 3-14, 2008.

[2] Begel A., Simon B., Struggles of new college graduates in their first software development job, *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, 226-230, 2008.

[3] Exter M., Comparing educational experiences and on-the-job needs of educational software designers, *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, 355-360, 2014.

[4] Hewner M., Guzdial M., What game developers look for in a new graduate: Interviews and surveys at one game company, *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 275-279, 2010.

[5] Misic M., Russo N., An assessment of systems analysis and design courses. *Journal of Systems and Software*, 45(3):197- 202, 1999.

[6] Nascimento, D., Almeida Bittencourt, R., and Chavez, C., Open source projects in software engineering education: A mapping study. *Computer Science Education*, 25(1):67-114, 2015.

[7] Nurkkala T., Brandle S., Software studio: Teaching professional software engineering. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 153-158, 2011.

[8] Radermacher A., Walia G., Gaps between industry expectations and the abilities of graduates. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 525-530, 2013.

[9] Spradley J., *The ethnographic interview*. Waveland Press, 2016.

[10] Strauss A., Corbin J., et al. *Basics of qualitative research*, volume 15. 1990.

[11] Sudol L., Jaspan C., Analyzing the strength of undergraduate misconceptions about software engineering. *Proceedings of the Sixth International Workshop on Computing Education Research*, 31-40, 2010.