Microservices and curricular education

Tullio Vardanega[1] and Riccardo Cardin[1]
[1] Department of Mathematics, University of Padua, Italy
tullio.vardanega@unipd.it, rcardin@math.unipd.it

Abstract

Frequently enough, one tends to think that his or her professional qualifications are solely the product of one's own work experience, void of traceable relation to curricular education. While this stance may reflect rightful frustration with curriculum instructors and teaching programs (as in "learning to read"), it is fundamentally flawed. Personal maturity, prerequisite of professionalism, rests on education and consolidates harmoniously with experience (as in "reading to learn"). Sure enough, breakages may well occur between curriculum design and professional competence, unfortunately. More frequently so where the knowledge domain evolves fast, faster than curriculum design may coalesce adaptively around it.

Where does the knowledge of microservices stand within the instructional design of university-level Computer Science curricula? This short paper reflects on this question, drawing from the experience of instructors deeply immersed in the professional practice.

1 Introduction

One of the challenges with the shaping of curricular education is to decide where to place the border between the learning outcomes of the curriculum design and the coverage that they should have of the advancement of the corresponding knowledge and state of practice. As science evolves, so does – with a variable lag – the state of practice. It stands to reason that the learning outcomes of curricular education should also advance accordingly.

Such an advancement, however, is not an easy matter, however, in several respects, which is where the challenge arises. The first and foremost difficulty is that a curriculum is a fixed-size container: if you want to fill it beyond capacity, all you will get is wasteful, if not outright disconcerting, overflow. In truth, what is fixed-size in the curriculum-container is the duration of the education cycle: the learning outcomes achieved across it might naturally evolve if the student's initial knowledge base progressed over time. In other words, a static zero-base of curricular education allows the instructional design to set in an equally static mode and content itself with transmissive teaching* only. That route is bound to cause the gap between curricular education and "the world out there" to widen frustratingly.

* Modern pedagogy categorizes teaching into transmissive, transactional, and transformative. Its body of knowledge recommends instructors to shift from left (transmissive) to right (transformative) in their instructional design.

Defeating this trend is not an easy endeavor. Success depends on a combination of factors: the instructor, the class scope, and the teaching institution.

This introduction, somewhat askew from the scope of this conference, serves the purpose of arguing that the thriving of the microservices paradigm in the state of practice depends *also* on the extent to which the associated knowledge enters the learning trajectory of curricular education. This short paper presents some reflections on that issue, concentrating on one of the three opportunities that instructors may encounter in a Computer Science curriculum at university level. Such three opportunities are: (1) software engineering projects for bachelor students, which may well be precursors to internships on the same theme; (2) exploratory assignments for master students; (3) end-of-study projects for master graduate candidates. Whereas opportunity (3) may occasionally be as rewarding as in (Vardanega & Simioni, 2018), and opportunity (2) may cause prerequisite interest to arise, this paper focuses on (1), which has the highest potential for harvest at the earliest possible exit of the education path.

2 Setting the scene: understanding microservices

Arguably, the microservices style is the tip of the history of software architecture and programming to date. It evolves from the initial monoliths, through the Service-Oriented Architectures (The Open Group, 2013), and is itself the precursor to the cutting-edge Serverless pattern (Fowler, 2018).

Teaching the rationale of the microservices architecture – over and above its technological incarnation – and what its concept entails is hard as it requires understanding that illness that it aims to cure. The hurdle is that this architectural style intimately relates to people's organization and relationships, which you must have experience of to appreciate properly.

Like any other software engineering pattern, the microservices architecture addresses a specific problem. For the microservices, it is the handling of dependencies among the components of enterprise applications. Often enough, such applications cover sizeable spans of the enterprise business processes, with large groups of heterogeneous professionals working on individual parts of them or on their overall aggregate. In such situations, the need arises to coordinate those work groups, which include software developers, business experts, database administrators, infrastructure managers, and other specialized profiles. The more transversal the application, the more diverse, composite and numerous the work group. Experience says, however, that there is a strict upper bound on how many people one can coordinate efficiently. Amazon CEO's rule of thumb, for one, is that, no matter how large your organization gets, individual teams should not be larger than what two pizzas can feed. The consequent best practice is to divide the application *and* the organization in small-sized groups, for easier management of both people and software components.

This ramification reflects the fact the microservices architecture concept very much relates to the Domain-Driven Design theory (Evans, 2010). More specifically, it is a punctual concretization of the notion of *bounded context*, perhaps the most effective means to date to tame organizational complexity.

3 Learning opportunities at CS bachelor level

3.1 Hazards

The first hurdle in exposing Computer Science bachelor students to microservices is that the preconditions for the use of such a structural practice to become measurably valuable simply cannot hold. There is pressure from the number of students who can take part in a collaborative educational project, which cannot grow freely. Likewise, there is pressure from the complexity of the domain model that the product of that project can have, which can only mirror reality from some distance.

For student groups of canonical size[†], it is highly likely that the same group (as opposed to separate sub-groups) will jointly develop *all* microservices. Unfortunately, engaging the same developers across distinct microservices shortcuts the critical aspects of their architectural composition, i.e., data and failure isolation, organization of persistence, communication. Not surprisingly, therefore, students tend to mistake the "distributed monolith", which is the frequent product of their eventual implementation, with a true microservices application. It is also not rare that the students initially fall in the anti-pattern of "nanoservices", legacy of their naïve understanding of Object Oriented Programming. That defect is comparatively easier for students to see and evade than the bigger issues at stake at architectural level.

In addition to those shortcomings, the students' learning of microservices in any such setting cannot possibly extend to production[‡]. Regrettably, this limitation keeps students away from experiencing the hardship of it, where microservices architectures expose the flank to fundamental vulnerabilities far more fiercely than they can do during development. API availability, performance bottlenecks, communication overheads, distributed transactions are the traits of a microservices architecture most liable to operational breakdowns when the system faces unexplored scenarios.

To an external observer of university education, this deficit might perhaps seem like a fundamental flaw in the instructional design of such projects. Yet, the whole body of literature in this field shows that there is no plausible room, for span of assignment, level of complexity, and – last but not least – attractiveness of challenge[§], to address that problem area in a single (horizontal) curriculum. Vertical internships may get nearer to operation, but only just, and always contingent on rare combinations of opportunity (at the host) and inclination (of the intern).

3.2 Learning outcomes

Setting expectations right is one of the most difficult tasks in the mandate of instructors. At present, the decision as to whether to expose students to microservices at some point in the bachelor curriculum is more frequently the privilege of a single instructor than a transversal priority of the faculty. In other words, it is an optional plus and not an obligatory must. Instructors using that privilege soon come to realize that the most plausible (and certainly valuable) learning outcome for students from it, is the familiarization with the technology challenge and solutions that underlie microservices architectures, possibly in relation with the principles of Continuous Delivery / Continuous Integration.

Setting goals much beyond it, is likely ill founded. More bluntly: expecting students to understand Domain-Driven Design (DDD) is foolishly erroneous. DDD is a truly complex field in itself, which can be approached soundly only with a profound understanding of the management of dependency between business and software components. For students who have just recently encountered the principles of Object Oriented Programming, and still struggle at scratching under the mere syntactic surface of it, it is unfair to pretend that they can have the sensitivity to understand how the DDD works at its heart, and divide a complex business model into smaller and well-defined *bounded contexts*.

4 Conclusions

The reader may have the impression that our drive in this short paper is to maintain that teaching microservices to bachelor students in a generalist Computer Science curriculum is inane. That is *not* our conclusion. Several parts of curricular education do not allow full and direct practice. They

[†] Common pedagogical practice suggests that the size of student teams for such project should be 6-7 peers, at the high end of Bezos' two pizzas.

[‡] Students do not eat their own food (a luxury very unlike professional life). In a university curriculum, project engagement necessarily ends with "product" release and never encounters real operation-and-maintenance issues.

[§] Human (and therefore student) nature is such that, in general, there is more creative pleasure in doing from scratch than in fixing the leftovers of others.

consequently lack an epiphany of understanding that be synchronous with classwork. The important thing, however, is that the corresponding learning mature in the students past their transition into adult life, and come to a deferred epiphany when the problem faced in the profession associates with situations evoked in classwork. One of the ingredients that favor deferred epiphany is linking class-time concepts with real-world situations, more or less patent or profound. Encountering and recognizing the latter will likely fire the association and cause epiphany. The ambit of microservices has many traits that make the link with real-world needs very apparent, and therefore warrants sowing the seed of it in the bachelor curriculum of Computer Science.

References

Evans, E. (2010). *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley.

Fowler, M. (2018, May 22). *Serverless Architectures.* Retrieved from martin.Fowler.com: https://martinfowler.com/articles/serverless.html

The Open Group. (2013). *What is SOA?* Retrieved from The SOA Source Boook: https://web.archive.org/web/20160819141303/http://opengroup.org/soa/source-book/soa/soa.htm

Vardanega, T., & Simioni, A. (2018). In Pursuit of Architectural Agility: Experimenting with Microservices. *IEEE International Conference on Services Computing, SCC 2018*, (pp. 113-120). San Francisco, CA.