

Edge Programming

Kevin Sullivan

University of Virginia Department of Computer Science
sullivan@cs.virginia.edu

Abstract

Complex systems of the future will often comprise networks of distributed programmable parts without centralized control over the programming of the parts. In other words, programming will happen at the edge. In this position paper I suggest that edge programming will create interesting and important problems for the field of software design and engineering. Chief among them will be to maintain conceptual integrity, thus ease of use, software quality, thus system dependability, and a coherent global view, thus system understandability and analyzability (at some level) in the absence of what to now has been a centralized control paradigm of software development. Edge programming is related to but distinct from end-user programming and from open-source development.

1. Some questions on edge programming

Ultra-large-scale systems will comprise networks of programmed parts without central control of software development. Software development and evolution will happen at the edges of such systems: in multiple, separate, decentralized units that manage their own software development processes. To capture this idea I suggest the term edge programming. The edges of complex systems are increasingly where the knowledge and other resources needed for effective decision-making will be located. Efficiency at the ultra-large scale demands decentralization. The software research community might profitably consider issues that will arise as we are forced to move from a model of software production based on tight central control to one involving decentralized complex adaptive systems.

Of course, edge programming is already here. For example, buying an item online activates many systems whose software is designed, developed, operated, and maintained by different organizations:

from the vendor to banks operating the payment system to the shipper to the buyer.

Although edge programming is not new, it has not been studied carefully as a distinct style of software or software-intensive system design and evolution. The practice outstrips our understanding. I suspect that edge programming, as a necessary style of ultra-large-scale system development, is likely to pose interesting and important problems to practitioners and researchers. Some of the questions that seem likely to arise include the following:

- How will conceptual integrity of the software in such a system, and thus its ease of use, be maintained, absent a centralized design team?
- How will the quality of the software deployed in a system, and thus dependability (including reliability, availability, security and safety), be assured, especially as the software in a system evolves without centralized quality control?
- How can an organization that devolves control of software design, development and evolution to its edge nevertheless maintain a useful global view of its software, e.g., for valuation, quality control, whole-system analysis, or certification by regulatory, insurance or related authorities?
- What principles should govern the design of infrastructure and policy to enable design, development, operation and evolution of high-quality, edge-programmed, software-intensive systems? A particularly important questions is what kind of infrastructure is needed to provide security and privacy, which are often functions of the weakest links in such systems?
- What principles should govern the design of an overall application system or function, and of activities within and across the organizations that design, develop, operate and maintain it?
- How does the edge programming model relate to end-user programming, to the open source model, and to other recent models of software development and evolution?

2. A few preliminary ideas

I have no definitive answers for most of the questions above. I can offer some educated speculation.

Conceptual integrity. Conceptual integrity will be maintained through open architectural standards. Such standards will, among other things, define the forms and minimally required functional and non-functional properties of *edge-programmable modules* from which such systems will be constructed. Of course this answer begs several key questions. Where will architectures come from? How will such architectures evolve? We can look to the evolution of today's standards (e.g., the PC, the internet protocol, etc) for insights.

Quality and dependability. Defining, achieving and maintaining such properties will pose significant challenges. Economic forces partly rooted in policy (e.g., defining financial or criminal liability for failures, outages, or harmful releases of information) seems likely to have an important role to play, consistent with devolution of responsibility to the edge. The feasibility of articulating and enforcing such policy, however, would appear to require advances in our ability to state required positive and negative properties of complex, software-intensive systems. A mixed approach might arise, with regulation of critical infrastructure services (information analogs of water or telephone service) but with markets alone regulating "higher-level" services. At a technical level, decentralized enforcement of key rules seems worth considering. Successful open source projects maintain tight centralized control over changes to the code base. Is there some way to automate and to decentralize this critical *admission control* function, e.g., to prevent operator of edge-nodes from installing new software that fails to pass certain quality tests, or that will allow them to install such software but only at the cost of reduced connectivity to the overall network?

Global view. Obtaining useful global views of very decentralized systems is essentially impossible today, if only because separate companies protect their software assets from outside inspection. The situation is perhaps more interesting within large organizations that have at least legal authority to create detailed global views. The question in this case is, by what mechanisms and to what ends? For example, how might a large national department of defense track and certify software parts developed or customized in the field? How can some level of coherence be maintained

if software is being created, customized, installed, configured and replaced at thousands of separate sites spanning ultra-large-scale command, control and intelligence and other systems?

Design principles. The idea of *edge programming* is meant to invoke the end-to-end principle that has been so important in the design of modern computer networks. Historically that idea has applied to decisions about *where to place functions* in distributed systems. The idea is that to the extent possible, rich functions, such as error correction, should be implemented at the edge, not in the core, because (a) such functions exact a cost, (b) that not all applications should have to pay, and (c) the designers of the core cannot adequately anticipate the needs of applications (the edges), and so should give them the tools to build what they need without unduly constraining them. The notion of *edge programming* embodies an analog applied to software development: the core defines minimal constraints and services while relegating most software development to the *edges* of the system. This notion is pretty much a polar opposite of accepted development models today.

Infrastructure design principles. An end-to-end principle in some form will plausibly emerge for designing ultra-large-scale information infrastructure systems of the future. That is, some core services and constraints will be provided. Everything else will be done at the edge. The question in my mind is what set of core services would suffice to promote development of a future society constructed largely around edge-programmed nodes? What kind of infrastructure, for example, will enable assurance that confidential patient information will not leak, even as many *thousands* of independently-programmed hospitals, pharmacies and insurance companies, large and small, are composed to provide a competitive and efficient ultra-large-scale drug dispensing function? What services might a core provide? What checkable constraints might it impose to ensure reasonable software development behavior within and across organizations involved in designing, developing, operating and maintain such a system?

Relation to other programming models. *Edge programming* as I envision it is distinct from a number of superficially related contemporary models. While *open source* uses networks to harness a decentralized labor force, it nevertheless employs centralized control over changes to code. Nor is edge programming (EP) equivalent to *end-user programming*. The emphasis in that research areas is not decentralized programming of complex systems, probably by professionals, but on enabling *non-technical* people to create software. Edge programming is related to the ideas of Prof. Mary Shaw and her students on *open resource coalitions*.

However, they assume essentially no regulation, and ask what can be done, while I ask what minimal regulation and infrastructure might be necessary to enable development through edge programming of robust systems on which society can significantly rely?

These issues seem worth exploring in this workshop.