The Future of Programming Languages and Programmers (Panel)

Steven Fraser

Principal Consultant, Innoxec
Innovation Executive Services, USA
sdfraser@acm.org

Lars Bak

Distinguished Engineer
Google, Denmark
lars@gunnestrup.dk

Rob DeLine

Principal Researcher
Microsoft Research, USA
Rob.DeLine@microsoft.com

Nick Feamster

Professor of Computer Science
Princeton University, USA
feamster@CS.Princeton.edu

Lindsey Kuper

Research Scientist
Intel Labs, USA
lindsey@composition.al

Crista Lopes

Professor of Informatics
University of California, Irvine, USA
crista@tagide.com

Peng Wu

Chief Scientist
Huawei USA
Peng.PengWu@huawei.com

Abstract

In the beginning "programs" were patterns of bits that commanded the execution of individual machines. As machines evolved in complexity – languages evolved, starting with a variety of assembly languages and growing to encompass higher levels of abstraction. Over the years – somewhat surprisingly – programmers evolved from engineers at the pinnacle of their profession with many years of experience to individuals not yet 10 years old giving evidence that programming does not necessarily require a formal education. This panel will bring together a diverse set of industry and academic professionals to discuss the future of programming languages and programmers.

Categories and Subject Descriptors:
D.3 Programming Languages
K.0 Computing Milieux
K.4.3 Organizational impact

General Terms: Languages

Keywords: Programming Languages, Programmers

SPLASH Companion'15, October 25–30, 2015, Pittsburgh, PA, USA
ACM. 978-1-4503-3722-9/15/10
http://dx.doi.org/10.1145/2814189.2818719

1. Steven Fraser

As Turing Award winner (1999) Fred Brooks described in his talk (79th minute of https://goo.gl/YXotkN) at the Computer History Museum's 40th anniversary celebration on the innovation of the OS/360 – it is possible for even brilliant people to create the "worst programming language ever designed." This was possible because the language designers didn't realize that they were in essence creating a language.

Increasingly, a growing diversity of programming languages are emerging to support and propagate innovation in our "always-on" and "always-connected" 21st century world. In turn these languages – some not likely discernible as "languages" by 20th century standards – have spawned a new generation of "programmers" who do not necessarily have a formal education as programmers or software engineers. Is

this a challenge – or feature – and what are the consequences for the future of programming languages and programmers?

STEVEN FRASER is the Silicon Valley Principal Consultant of Innoxec (Innovation Executive Services) advising on strategic innovation and technology transfer. From 2007 to 2013, Steven was the Director of the Cisco Research Center where he increased the visibility and leverage of Cisco-university research collaborations, fostered technology transfer from university research projects through the recruitment of PhD/Post-Docs, and accelerated internal technology transfer through the establishment of the Cisco Research Commons and the CTech Forum. Prior to joining Cisco, Steven was a Senior Staff member of Qualcomm's Learning Center in San Diego where he led software learning programs and created the corporation's internal technical conference (the QTech Forum) sponsored by the CTO and CEO. Late in the last century, Steven held a variety of technology strategy roles at BNR/Nortel including Senior Manager (Disruptive Technology and Global External Research) and Advisor (Design Process Engineering). In 1994, he consulted as a Visiting Scientist at the Software Engineering Institute (SEI) on the "Application of Software Models" project with the development of team-based domain analysis techniques. Steven received his doctorate in Electrical (Software) Engineering from McGill University in Montréal and holds a MS in Engineering Physics from Queens University in Kingston and a BS in Physics and Computer Science from McGill. Steven is a Senior Member of both the ACM and the IEEE.

2. Lars Bak

Companies and developers are unfortunately fairly conservative when deciding which programming languages to use. Sadly, universities set the standard by teaching mostly traditional languages like C, C++, Java, and JavaScript. This means that a new programming language only has a chance of succeeding if it radically improves programmer productivity in a certain domain.

The rapid growing IoT market might be such a domain that warrants a new programming platform. IoT devices have limited hardware capabilities, are network connected, and are most often battery powered. Today, an IoT device is likely programmed by statically linking the entire software stack into a binary and then re-flashed onto the device resulting in long edit-compile-run cycles and impossible to upgrade software. Programming of IoT devises can be vastly improved by supporting:

- An isolate/actor model with secure execution that safely allows installation of third party services.
- Incremental execution for fast development cycles.
- Incremental updating of software components.
- Predictable memory and CPU usage.

A new programming language will not alone make or break the success of such a new platform. Its success is equally dependent on solid libraries, a performant implementation, and an efficient programming environment. I am hopeful that the IoT market will inspire new programming platforms that in turn will improve programmers' productivity.

LARS BAK is a software engineer at Google and a veteran virtual machinist. He co-designed the Dart programming language with Kasper Lund and has spent the last five years making Dart an effective programming platform. His passion for designing and implementing object-oriented programming languages started in 1986 when implementing a runtime system for Beta. Since then, Lars has left marks on several software systems: Self, Strongtalk, JVM HotSpot, JVM CLDC HI, OOVM Smalltalk, V8, and lately Dart. Lars graduated from Aarhus University in 1988 with a MS degree in computer science.

3. Rob DeLine

In many cases, a new programming language becomes popular not because of its inherent qualities, but because it allows entry into an intriguing new programming domain. Certainly no one would argue that Javascript, PHP, or R are beautiful, usable languages (quite the contrary!), but for early adopters of client-side scripting, server-side scripting, and data science, these languages were the only game in town. Programming languages that arrive second in a domain struggle for adoption, even when they are technically superior. For better or worse, to get traction, the late arrivals often have to accommodate the design choices of the entrenched early languages. For example, C++ and Objective C kept C's execution model, and Python is reusing R's data organization concepts. In short, if you want to invent the next major programming language and particularly if you want the freedom to make all the design choices, you should choose (or invent!) a programming domain with no existing programming support, then create a language for it. Some examples could be augmented reality, wearables, e-health, and drones.

Luckily, there are also new opportunities for understanding how people actually use programming languages as design objects. For example, the vast repositories of open-source projects, like Github, provide a window into how features are used in practice and how updates to languages and runtimes are adopted over time. Researchers are also beginning to use biometric data, like eye gaze, pupil dilation, skin response and brain scans, to understand at an anatomical level how programmers interact with code. The promise is that we can have real-time signals for confusion, frustration, insight and other responses, so we can settle debates about programming language design with data, rather than endless opinionated debate.

ROB DELINE, a Principal Researcher at Microsoft Research, has spent the last twenty-five years designing programming environments for a variety of audiences: end users making 3D environments (Alice); software architects composing systems (Unicon); professional programmers exploring unfamiliar code (Code Thumbnails, Code Canvas, Debugger Canvas); and, most recently, data scientists analyzing streaming data (Tempe). He is a strong advocate of user-centered design and founded a research group applying that approach to software development tools. This approach aims for a virtuous cycle: conducting empirical studies to understand software development practices; inventing technologies that aim to improve those practices; and then deploying these technologies to test whether they actually do.

4. Nick Feamster

Communications networks remain incredibly difficult to manage, troubleshoot, and secure. Software Defined Networking (SDN) is a new approach which decouples the logical network control from the underlying network infrastructure. SDN will simplify many network management tasks in different types of networks and may ultimately provide a means by which network operators (and home users) can make their networks more predictable, manageable, and secure. We have developed and deployed "Kinetic" a new programming language and runtime environment for SDN home networks and large campus networks. Kinetic enables network operators to express and implement complex policies in a simple and high-level control framework. Current SDN controller platforms typically offer little domain-specific support for programming changes to data-plane policy over time (dynamic policy). As a "networking" person I know very little about programming languages research – however I foresee the day when we will all at some level be "programmers".

NICK FEAMSTER is a professor in the Computer Science Department at Princeton University and the Acting Director of the Center for Information Technology Policy at Princeton University. Before joining the faculty at Princeton, he was a professor in the School of Computer Science at Georgia Tech. He received his Ph.D. in Computer science from MIT in 2005, and his S.B. and M. Eng. Degrees in Electrical Engineering and Computer Science from MIT in 2000 and 2001, respectively. His research focuses on many aspects of computer networking and networked systems, with a focus on network operations, network security, and censorship-resistant communication systems. In December 2008, he received the Presidential Early Career Award for Scientists and Engineers (PECASE) for his contributions to cybersecurity, notably spam filtering. His honors include the Technology Review 35 "Top Young Innovators Under 35" award, the ACM SIGCOMM Rising Star Award, a Sloan Research Fellowship, the NSF CAREER award, the IBM Faculty Fellowship, the IRTF Applied Networking Research Prize, and award papers at the SIGCOMM Internet Measurement Conference (measuring Web performance bottlenecks), SIGCOMM (network-level behavior of spammers), the NSDI conference (fault detection in router configuration), Usenix Security (circumventing web censorship using Infranet), and Usenix Security (web cookie analysis).

5. Lindsey Kuper

The moral of Fred Brooks' story about "the worst programming language ever designed" is that it behooves us as programmers to approach certain problems as language problems from the start, since those problems will end up being language problems anyway. To that end, I would like to see a future in which all programmers are empowered to think of ourselves as language designers and implementors. By that, I do not mean that we all necessarily ought to go out and start our own large-scale language projects from scratch. Rather, I mean that we should teach programmers to appreciate how the implementation of every language is ultimately just another program, written by other humans; that we should gravitate toward languages that are easily extensible, and toward languages that allow and reward messing with their inner workings; and that we should be thoughtfully critical of the tools we have, and of new tools that come along, rather than merely accepting them as they are handed to us.

Is programming more accessible today than in the past? First of all, it depends on how one defines "the past"; in the 1940s, programming was seen as clerical work, and it wasn't until about fifteen years later that it began to be taken seriously as an engineering discipline. Starting around 1980, programmable microcomputers became a mass-market product, which did good things for the accessibility of programming. But today, although mass-market computers are cheaper and more ubiquitous than ever, most of the interesting computation is done in the cloud, on large-scale, institutionally-owned hardware.

The challenges I see for programming effectively in such a world include dealing with quantities of data too large to fit on any single machine, and reconsidering old assumptions about the consistency of network connections or durability of storage media. On this panel, I look forward to discussing how the future of programming could address those challenges, and which other challenges we might be neglecting to take seriously today that will be important in another fifteen years.

LINDSEY KUPER is a Research Scientist in the Programming Systems Lab at Intel Labs. Prior to joining Intel last year, she did her dissertation work on LVars: lattice-based data structures for deterministic parallel and distributed programming. Her open source software contributions have included work on Parallel JavaScript at Intel Labs and on the

Rust programming language at Mozilla Research. In 2014, she co-founded !!Con (http://bangbangcon.com), a conference about the joy of programming. She is interested in tools, techniques, and abstractions that support compositional reasoning about programs, proofs, and processes, especially those that don't immediately appear to play nicely together. She holds a Ph.D. in computer science from Indiana University and a BA in computer science and music from Grinnell College.

6. Crista Lopes

There are two well-known sides to programming: (1) mastery of the domain; and (2) mastery of the formalisms by which intentions become executable by computers (we call these programming languages).

There is also a third side: the mastery of the computing systems that run the programs. This third side has been treated as a red-haired step-child by programming language designers, researchers and even educators; much of the effort towards high(er) level languages has been to hide the complexities of computers from the people who have the domain knowledge. That works, but only up to a certain extent. The operational environment of programs establishes many non-functional requirements that have become the blind spot of software research, from their expression to their verification and testing. It is important to analyze the extent to which hiding operational complexity is beneficial and the point at which it becomes an obstacle.

CRISTA LOPES is a Professor of Informatics at the University of California, Irvine. Her research focuses on software engineering for large-scale data and systems. Early in her career, she was a founding member of the team at Xerox PARC that developed Aspect-Oriented Programming. More recently, she has been focusing on understanding how people write code and use programming languages by mining large software repositories. Along with her research program, she is also a prolific software developer. Her open source contributions include being one of the core developers of OpenSimulator, a virtual world server. She is also a founder of Encitra, a company specializing in online virtual reality for early-stage sustainable urban redevelopment projects. She has a PhD from Northeastern University, and MS and BS degrees from Instituto Superior Tecnico in Portugal. She is the recipient of several National Science Foundation grants, including a prestigious CAREER Award. She claims to be the only person in the world who is both an ACM Distinguished Scientist and Ohloh Kudos Rank 9. She authored Exercises in Programming Style, a book that has been gaining rave reviews among software aficionados.

7. Peng Wu

The future of programming is driven by three important shifts in the "what", "how", and "who" of programming:

1. "What" can be programmed? Almost everything can be programmed. We will (and have already) see(n) a dramatic surge on the varieties of "things" (hardware devices) that are programmable, from the entire network (so-called SDN, Software-Defined-Network), to tiny sensors (so-called IoT), or maybe your "smart" refrigerator at home.
2. "How" programming is done? Programming is moving from the backstage to the spotlight and becomes a part of the "customer experience". As software services become increasingly layered and complex, an increasing number of software products empower their customers to program the service via programmable platforms and services.
3. "Who" are the programmers? Almost everyone can be a programmer, and a lot of them are not of Computer Science major. Some can be as young as 10 years-old. Others are domain expertise who are simply using programming to help solve their technical problems.

As the target hardware (what), business model (how), and demographics (who) of programming changes, we'll see the future of programming will likely focus more on user experiences (e.g., ease-of-use), unconventional natural programming interfaces, intelligent tools, and at the same time applications will be highly specialized (domain-specific).

PENG WU recently joined Huawei US research lab and help founded the Programming Technologies Lab at the Central Software Institute of Huawei. Serving as the first director of the Programming Technologies lab, her mission is to leverage programming technologies (languages, VMs, and tools) to improve the software capability of Huawei, a global telecommunication company that is redefining herself in the era of ICT (Information and Communication Technology) convergence. Before joining Huawei, Peng Wu was a research staff member at IBM T.J. Watson research center for more than a decade and has done extensive research in SIMD programming, compilers, scripting languages, TM, and software-hardware co-design. Peng Wu received her PhD in computer science from University of Illinois, Urbana-Champaign in 2001. She continues to be actively engaged in the research community and has co-authored more than 30 papers (including a best paper award in PACT'12) and held more than 20 patents. She is an adjunct professor in the Department of Computer Science at University of Illinois, Urbana-Champaign since 2012. For more information, check out http://pengwu.wordpress.com