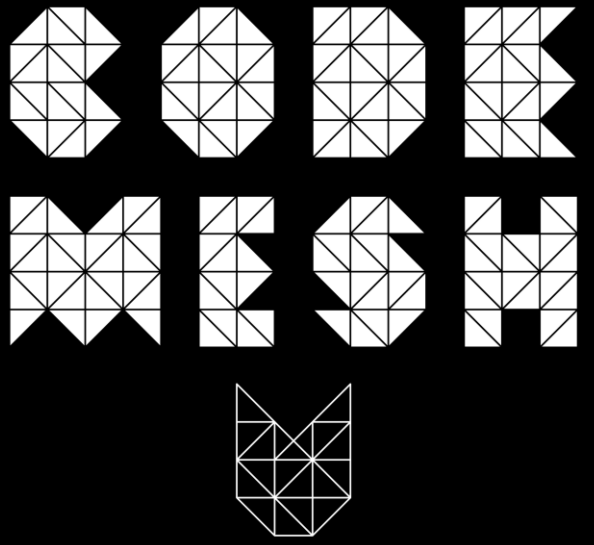
# Cultures of Programming

## Lessons from 70 years of learning how to control the electronic computer

**Tomas Petricek**, University of Kent
tomas@tomasp.net | @tomaspetricek



# Structure of Scientific Revolutions (1962)

What differentiated these various schools was not one or another failure of method—they were all ‘scientific’—but what we shall come to call their incommensurable ways of seeing the world and of practicing science in it.

Thomas Kuhn

# Different ways of programming?

How to specify programs?

How to avoid program errors?

What are types good for?

How to make sure it works?

What is object-orientation?

What activity is programming?

Hacker culture

Mathematical culture

Business culture

Engineering culture

Creative culture

# Scene 1

## ENIAC and EDSAC: Programming in the 1940s

T It is 1946 and the ENIAC computer has just been introduced. This raises a question of how to program such machines?

B Surely, building a computer can be outsourced to a sub-contractor like work on other scientific instruments!

H This is hard… You have to fiddle with the machine and try things out!

M I think you should figure things out logically before trying.

T Did anything change in 1949 when the EDSAC computer was built?

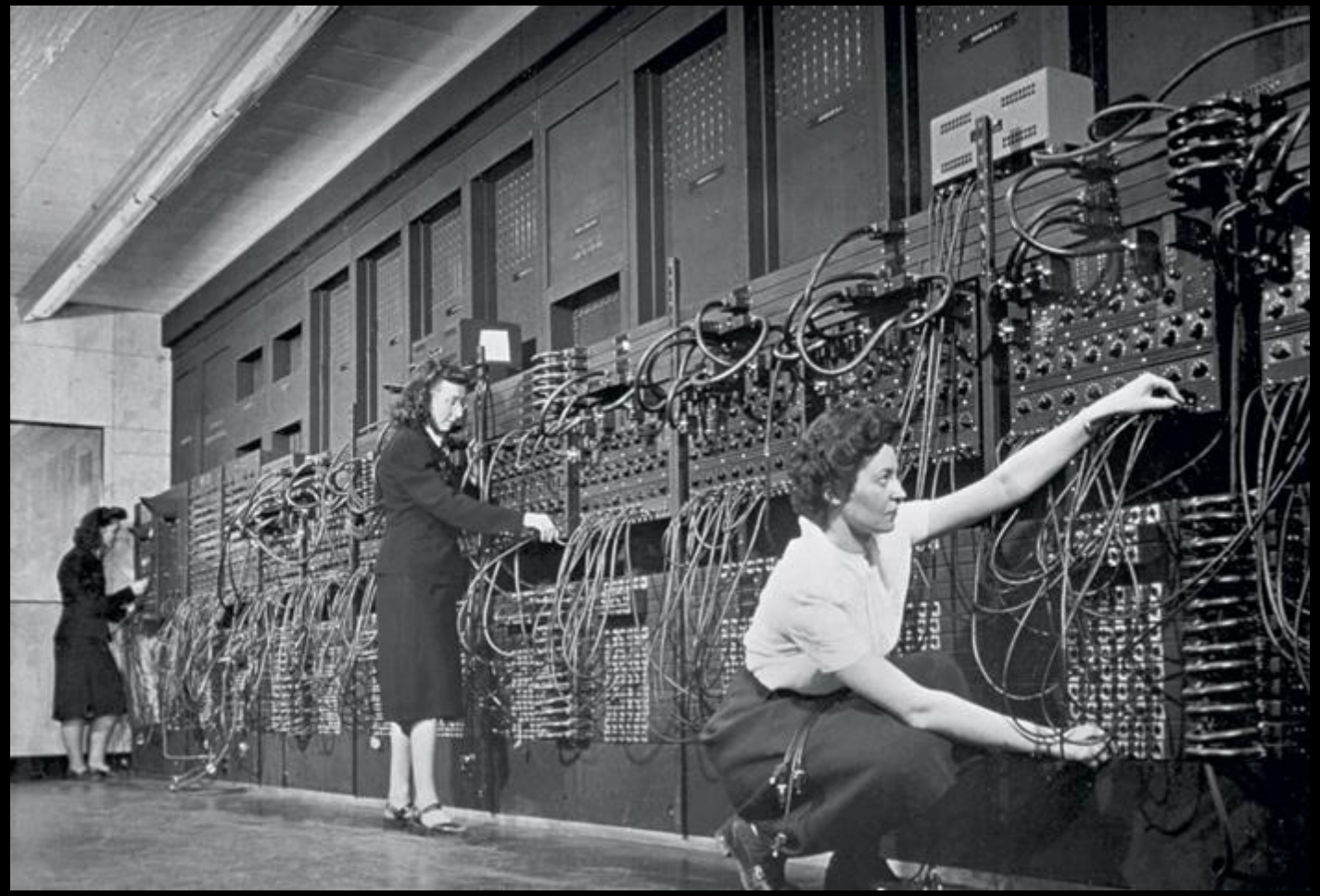
H It’s still the same, but punched cards make it a bit quicker to try things out!

E Oh, but we could reduce the number of instructions by introducing subroutines!

B Good, because this fiddling is expensive. Programs should be run by operators to use the machine effectively.

“The ENIAC was a son of a bitch to program”

**Betty Jean Jennings Bartik**

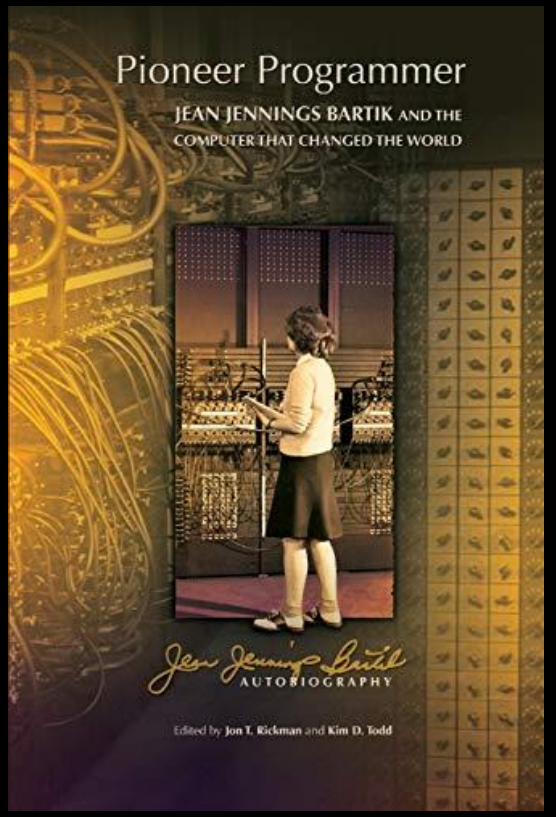


# Debugging the ENIAC

The ENIAC had (...) five buttons, which read "Start," "Stop," "Continuous," (...) We could [also] disconnect one of the program-pulse output cables and stop it after a particular operation.

Many times we could not tell whether the ENIAC was making an error [because of a blown tube] or we had a bug in our program.

Betty Jean Jennings Bartik



# Hackers and Mathematicians

Adele [Goldstine] was an active type of programmer, trying things very quickly. I was more laid back and given to attempting to figure out things logically before doing anything.

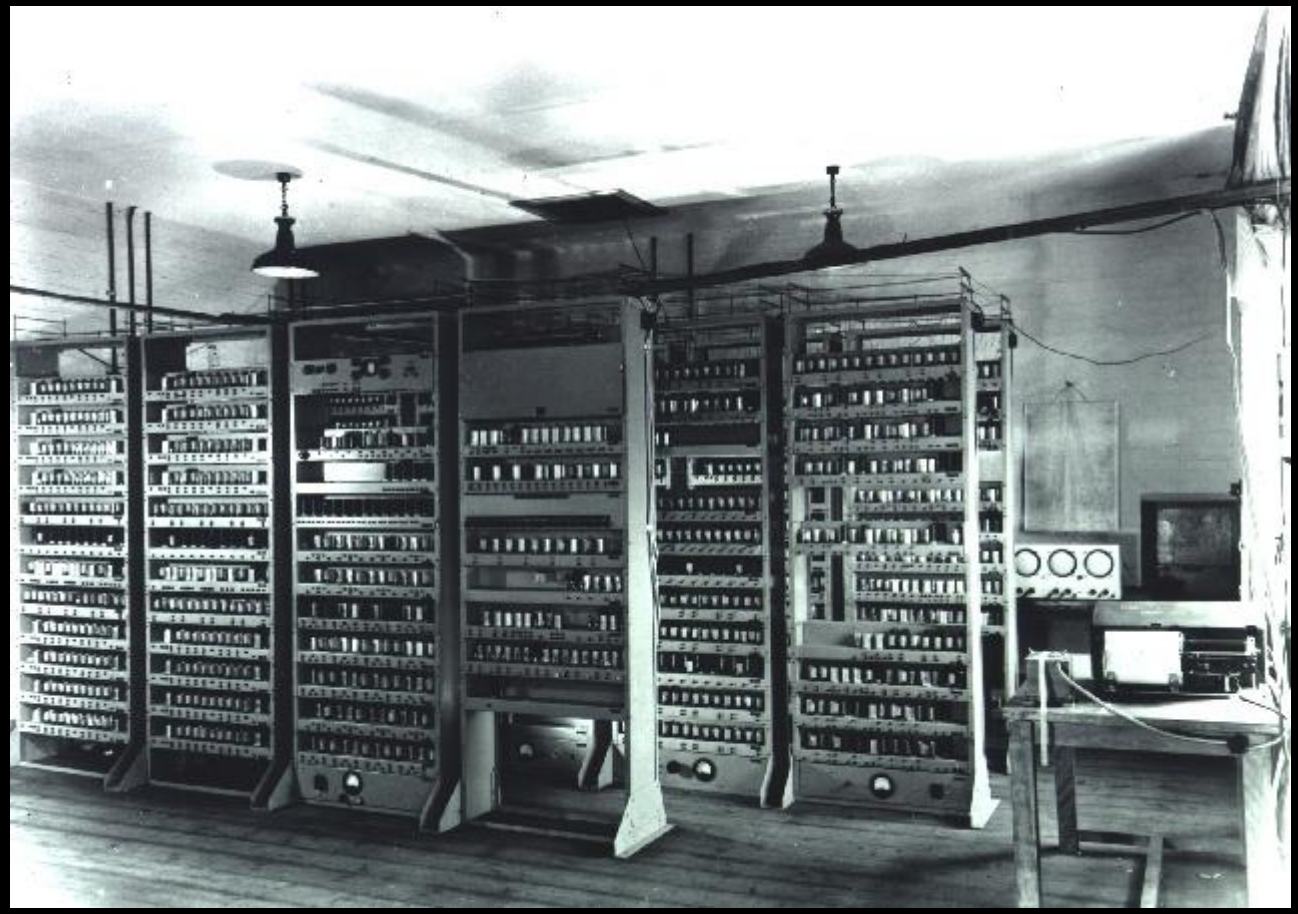
Betty Jean Jennings Bartik

# Programming the EDSAC

Completed in 1949
in Cambridge, UK

Designed as a stored
program computer

Programmed using
punched cards



# Debugging the EDSAC

The way to debug a program (…), was to sit at the console and execute the program manually, instruction by instruction, while observing the registers and memory on monitor tubes (…).

Martin Campbell-Kelly (2011)

# Debugging the EDSAC

**Post-mortem routine** – print out a region of the store so that it can be studied offline if program gores wrong.

**Checking routine** –program executed by an "interpreter" that printed diagnostic information

"From early 1950, it became possible to assign a full-time operator to the EDSAC who would run programs on behalf of users."

Martin Campbell-Kelly (2011)

**From Theory to Practice: The Invention of Programming, 1947-51**

Martin Campbell-Kelly

Warwick University

**Abstract.** This paper describes the development of programming for the EDSAC computer at Cambridge University, beginning in 1948 and culminating in the publication in 1951 of the classic *Preparation of Programs for an Electronic Digital Computer* by Maurice Wilkes, David Wheeler and Stanley Gill. The relationship to earlier programming studies conducted by Herman Goldstine and John von Neumann during 1947-1948 at the Institute for Advanced Study, Princeton University, is discussed. The subsequent diffusion of the Cambridge programming system and its influence are described.

**Prolog**

It is difficult to reconstruct how one came to choose a direction in life, a particular degree subject or profession. I think this is because big decisions are the result of many small events, some more memorable that others. For me, the most significant event putting me on the road to computer history occurred when I was cramming for my finals as an undergraduate in computer science at Manchester University in 1969. I was working in the Radcliffe Library and happened to come across an obscure, to me, book called *The Preparation of Programs for and Electronic Digital Computer* by Wilkes, Wheeler and Gill (1951). The book was about the programming regimen devised for the Cambridge University EDSAC. It was rather like looking at computing from another civilization, almost another planet. The book was clearly about programming, but not as I knew it. Well—my finals had to take priority and I re-shelved the book. But I never forgot it.

A few years passed and I eventually ended up as a senior lecturer at what was then Sunderland Polytechnic. In those days—it was 1976—computer science graduates were thin on the ground, so it was possible to get a job in an academic department without a higher degree. The Polytechnic, which had aspirations to improve its academic standing, had the enlightened policy of encouraging staff to pursue a PhD degree on a part-time basis. I leapt at the opportunity, and after some deliberation about whether I should do something relevant I decided to pursue the irrelevant—a PhD in the history of computing. It turned out to be not irrelevant at all.

But first, I needed a supervisor. By a life-changing stroke of luck, Newcastle University was just 15 miles up the road from Sunderland, where Professor Brian Randell had recently published *The Origins of Digital Computers* (1973). The book was an instant classic and I recognized it as such. I was quite nervous about approaching Brian, but he invited me for a talk and instantly put me at my ease. We agreed almost immediately that a study of the early development of programming

C.B. Jones and J.L. Lloyd (Eds.): Festschrift Randell, LNCS 6875, pp. 23–37, 2011.

**Hacker culture**

Direct interaction with the machine, often using individual expertise.

**Mathematical culture**

Treat programming as mathematical challenge to be solved logically.

**Engineering culture**

Use the machine to build better tools for controlling, testing and debugging.

**Business culture**

Computers as a scientific instrument. Human processes to use it efficiently.

# Scene 2

Time-sharing and minicomputers in 1950-1960s

T It is 1956 and programming is done via batch processing, with the exception of experimental machines like TX-0. Is this the start of a computing revolution?

H Absolutely. You can even modify program while sitting at the computer. A miracle!

B Blocking a $3 million machine for hours? What a waste of resources.

E Maybe we could enable the same interaction by letting multiple users work concurrently with a single machine?

C Imagine if you could use computer like this in a classroom and use it to teach children about thinking!

H Time-sharing may work fine for that, but for real interactive use that uses screen rather than terminal, computers need to become personal.

# Programming TX family computers

Users debugged their programs right at the console, sitting there sometimes for  hours (...) usually at night.

Using these features (...) Ivan Sutherland  was constructing [Sketchpad] a system that displayed drawings with which users could interact (...) in real-time.

Severo Ornstein (2002)

# But interactive computing is expensive!

The **Big Dealers** solution was to divide up the machine's cycles in such a way that many [remotely connected] users could use it at the same time.

**Small Dealers** felt that real-time, interactive use via a display was crucial and that Time-Sharing would never provide such capability.

Severo Ornstein (2002)

# Creative culture enters the scene in 1960s

**Sketchpad** makes it possible for a man and a computer to converse rapidly through the medium of line drawings.

Ivan Sutherland (1963)

The **LOGO** classroom experience was a revelation! This was more like the environment of powerful epistemology, the environment of media.

Alan Kay (2013)

# Business culture strikes back in 1970s

**VisiCalc** was one of two application products (with WordStar) that were (...) really the software underpinnings for the (...) explosive growth of the personal computer industry.

Burton Grad (2007)

**Hacker culture**

Explore computers for the sake of computers, requiring full control.

**Business culture**

Focus on business problems affects what interaction is supported.

**Engineering culture**

Again, solving computer-related problems by using the computer.

**Creative culture**

Use computers as new media and in unexpected ways, often in education.

# Interlude

## Why cultures of programming matter

# Why cultures of programming

**Past**

Make sense of past developments!

Cultures clash as well as collaborate.

**Present**

Combine ways of problem solving.

Avoid cultural misunderstandings!

**Past**

Imagine unexplored new developments.

Know where to look for cool ideas.

# Scene 3

## What kind of activity is programming?

T The very idea of a programming *language* is a metaphor. Is this just an accident?

M Not an accident. To program is to manipulate terms in formal languages!

H It is just bits in memory!

C Children programming Turtle with LOGO do not think of formal languages. Programming is interacting – with the medium.

B Does that mean users can arbitrarily change your programs!?

E Even if programming is using a language, you can build various useful tools…

T Is it decided then? Is programming like writing?

C All creative uses of programming are interactive!
But there is more – think about data science tooling.

# When technology became language

How it became natural to think of programming as a linguistic activity?”

From “programming notations as attributes of individual machines” to free-standing notations, drawing “on the disciplines of symbolic logic and linguistics” for their description.

Nofre et al. (2014)

When Technology Became Language

The Origins of the Linguistic Conception of Computer Programming, 1950–1960

DAVID NOFRE, MARK PRIESTLEY, and GERARD ALBERTS

Introduction

The second half of the 1950s saw the emergence of a new vision of how computers were to be programmed. At the beginning of the decade, programmers had to express the instructions for solving a problem in obscure numerical codes that were different for each machine. By the decade's end, however, they could write programs that included familiar mathematical formulas, and, in some cases, even expect the same program to run on different machines, thanks to the development of systems like FORTRAN and IT. Furthermore, professional and industrial bodies were putting forward ambitious proposals for very powerful “programming languages,” as the codes were now widely called, and some of these, notably ALGOL and COBOL, were explicitly defined to be machine-independent notations. In

David Nofre is associated with the Centre d'Estudis d'Història de la Ciència, Universitat Autònoma de Barcelona; Mark Priestley is an independent researcher based in London; and Gerard Alberts is an associate professor of the history of mathematics and computing at the Korteweg-de Vries Institute for Mathematics, University of Amsterdam. Nofre and Alberts's contribution was developed as an element of the Software for Europe project, as part of the European Science Foundation Eurocores Program “Inventing Europe,” and co-funded by the Netherlands Organization for Scientific Research (NWO 231-53-004). Research for this article was assisted by the award to Nofre of a 2010 Lemelson Center Travel to Collections Award from the Smithsonian Institution and a 2009 Arthur L. Norberg Travel Award from the Charles Babbage Institute. The authors thank Eden Medina for helpful comments on an early draft of this article; Matthias Dörries, Helena Durnová, Hans Dieter Hellige, Janet Martin-Nielsen, and Edgar Daylight for insightful comments on its early ideas; and the three anonymous referees and Suzanne Moon for providing constructive comments and suggestions. They also thank Peggy Aldrich Kidwell for access to materials in the Computer Documentation Collection at the National Museum of American History, Smithsonian Institution, Washington, D.C.


0040-165X/14/5501-0002/40–75

# Creative origins of Smalltalk

“The purpose of the Smalltalk project is to provide computer support for the creative spirit in everyone.”

“If a system is to serve the creative spirit, it must be entirely comprehensible to a single individual.”

Dan Ingalls (1981)

# Interacting with Smalltalk

“A user interface is simply a language in which most of the communication is visual.”

“Every component accessible to the user should (...) present itself in a meaningful way for observation and manipulation.”

Dan Ingalls (1981)

# What killed Smalltalk?

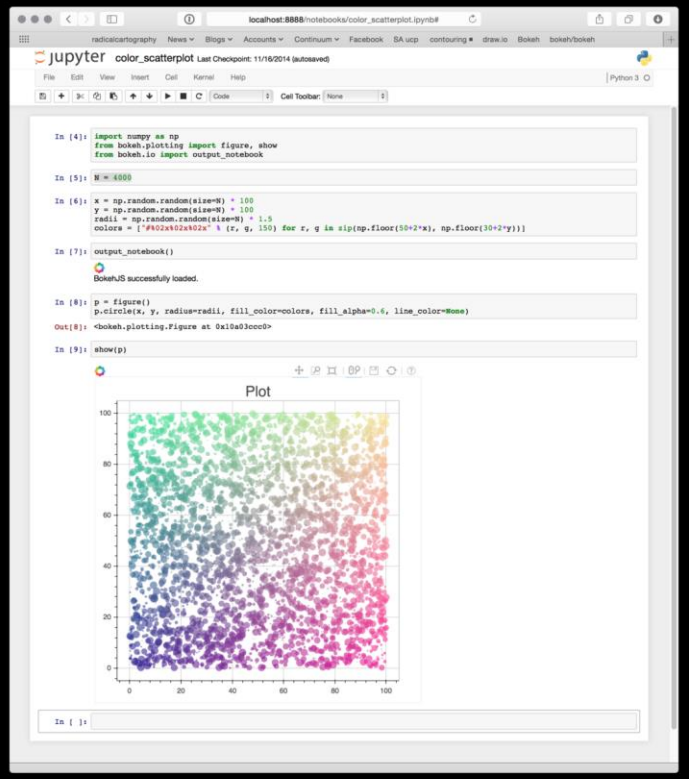
**Lack of a standard.** “Each vendor had a slightly different version - not so much a different language, as a different platform.”

**Deployment.** “Smalltalk (…) computation takes place [in] a ‘sea of objects’. If you want to deploy an application by separating it from the IDE (to (…) protect your IP) it turns out to be very hard.

Gilad Bracha (2020)

# Where programming is still interacting?


Live coded music




Exploratory data analytics

**Mathematical culture**

Programming as a formal language works well for academic research!

**Business culture**

Programs as closed artifacts works well in the commercial setting.

**Creative culture**

Programs as interaction with dynamic medium allows more creative uses!

**Hacker culture**

Programs lists of bits let you see what actually goes on in a computer.

# Conclusions

## Making sense of programming past & future

# Cultures of Programming

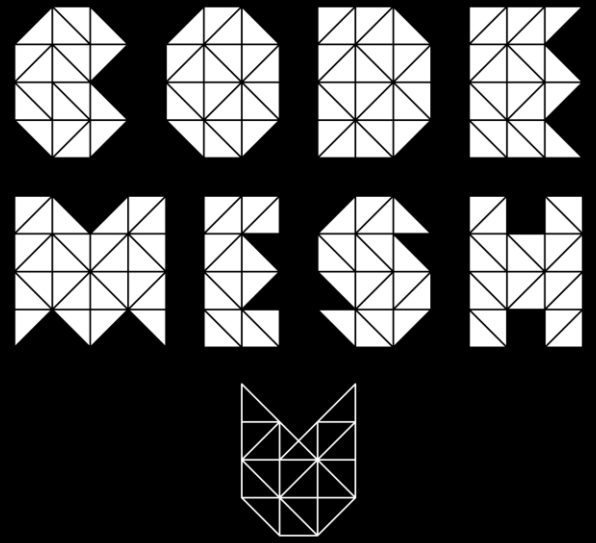
**Are here to stay.** After 70 years, it's not just a sign of an immature field.

**Interact in many ways.** Clash over principles, collaborate to improve programming practice.

**Help us understand.** Controversies of the past, current debates on hiring, methodologies, etc.

# Thank you!

Mathematical, engineering, hacker,
business and creative cultures
to make sense of programming!



**Tomas Petricek**, University of Kent
tomas@tomasp.net | @tomaspetricek