The Development and Deployment of Formal Methods in the UK

CLIFF B. JONES, School of Computing, Newcastle University
MARTYN THOMAS, Emeritus Professor of IT, Gresham College

In addition to the major UK contributions to research underpinning formal approaches to the specification and development of computer systems—and perhaps as a consequence of this—some significant attempts to deploy the ideas into practical environments have taken place in the United Kingdom. The authors of this article have been involved in formal methods for many years and both had contact with a significant proportion of this history. This article both lists key ideas and indicates where attempts were made to use the ideas in practice. Not all of these deployment stories have been a complete success and an attempt is made to tease out lessons that influence the probability of successful long-term changes to software engineering.

CCS Concepts: • **Software and its engineering → Software notations and tools**; **Semantics**; **Software creation and management**; **Software development techniques**; **Software verification and validation**; **Formal software verification**;

Additional Key Words and Phrases: Industrial use of formal methods, VDM, Z, CSP, CCS

ACM Reference format:
Cliff B. Jones and Martyn Thomas. 2022. The Development and Deployment of Formal Methods in the UK. *Form. Asp. Comput.* 34, 1, Article 6 (July 2022), 21 pages.
https://doi.org/10.1145/3522577

1 INTRODUCTION

The term "formal methods" covers the use of mathematically precise notations to specify and to reason about systems. This article is mainly concerned with formal methods for software development. As well as mentioning some of the important scientific insights underlying such methods, emphasis is placed on the application and deployment of technical ideas and their tool support.

Both of the authors of this article believe that the use of formal methods is essential for the creation of high-quality software. This article is, however, not a sales pitch for such methods. What it sets out to do is to review some actual deployments, to be frank about drawbacks as well as report successes and to identify factors that have either aided or impeded the deployments.

There is no claim that this article reports on all (nor even the most important) applications of formal methods; [34, 102] (and their citations) offer broader surveys, [33] presents a balanced

Jones gratefully acknowledges the support for his research of grant RPG-2019-020 from the Leverhulme Foundation and the EPSRC Strata Platform Grant.
Authors' addresses: C. B. Jones, School of Computing, 1 Science Square, University of Newcastle, Newcastle upon Tyne, NE5 5TG, UK; M. Thomas, Gresham College, Barnard's Inn Hall, Barnards Inn, London EC1N 2HH, UK.

0934-5043/2022/07-ART6 $15.00
https://doi.org/10.1145/3522577

SWOT analysis; and some recent relevant publications are indicated in Section 6.2. Given that a selection had to be made, it felt reasonable to choose applications where the authors had personal knowledge or at least had direct access to colleagues with such knowledge. There is thus a strong UK emphasis, which came about initially because the article was submitted to another journal in which there was a plan to have companion articles covering other countries. The personal contact reveals factors that cleaned-up success stories might omit and we attempt to draw lessons from these factors.

There is no attempt here to claim that formal approaches are a panacea in the sense that they would solve all of the problems in the software industry. For example, the question of how to argue that a (formal or informal) specification actually meets the needs of users is a huge challenge—the wise writings of Michael Jackson such as [51] are a good starting point on this topic. Furthermore, there is the challenge of designing systems that are dependable when used in conjunction with faulty components (including hardware and human)—in this area [12, 86] are useful starting points.

Beginning with the theoretical insights, Tony Hoare's article on the axiomatic basis of programming languages [46] is a foundation stone of a significant portion of the work on formal methods.[1] In tandem with notations that have been used to provide abstract models of complex systems (e.g., **Vienna Development Method (VDM)** [55], Z [41], B [1], and Event-B [2]) Hoare's ideas permeate many of the applications discussed in the body of this article. Robin Milner's LCF [36] system provided a model for many of the theorem proving assistants that have been developed.[2] UK computer scientists have also made major contributions to concurrency research (e.g., Milner's CCS [72] and Hoare's CSP [47]).

Approaches to recording the formal semantics of programming languages is another area where research in the United Kingdom has made crucial contributions; although this plays a small part in the rest of the article, Section 2 outlines this research and identifies some of its applications.

We decided against trying to follow a strict timeline for the article. This is partly because the various strands of the story overlap heavily in time, but we also felt that the lessons could be more clearly illustrated by looking at different modes of engagement. Beyond Section 2, the structure of the article is that Section 3 considers initiating academic/industry collaboration using consultants, Section 4 reports on some projects where the expertise was in-house, Section 5 identifies some factors that have an impact on any deployment and finally conclusions are summarised in Section 6.

2 PROGRAMMING LANGUAGES

Although the bulk of this article is concerned with the specification and development of programs, there are two reasons to address research on programming languages: (i) This was one of the first areas where it was realised that formalism could make a contribution, and (ii) UK researchers played a significant part in the development of such research.

The study of languages is sometimes referred to as "semiotics".[3] For programming languages, the most important facets are syntax (covering content and structure) and semantics (or meaning). Finding suitable notations to define the syntax of programming languages was achieved both early and with broad consensus. The ALGOL 60 language was described in [9] using ***Backus Normal Form(BNF)***[4] (often referred to as *Backus Naur Form*).[5] Variants of BNF have been developed

[1] An excellent technical survey of 50 years of research on *Hoare-Logic* is [5]. An attempt to outline the broader picture is [57].

[2] Historical articles on these topics include [35, 65, 73, 80] looks at what it means to claim that something has been proven.

[3] Charles Sanders Peirce 1839–1914 wrote extensively on philosophy, language and logic—in addition to his collected works, [81] reprints an earlier book.

[4] John Backus made the initial proposal.

[5] Peter Naur applied the notation to ALGOL and proposed extensions to the notation.

including *Extended BNF* [98] and, also from Niklaus Wirth, a graphical representation referred to as *Railroad Diagrams*. All of these notations perform essentially the same function and there is little doubt as to their usefulness. One bonus from using such formal descriptions of the syntax of programming languages is that they can be employed to generate parsers for the front-ends of language translators. This does bring some additional concerns about ambiguity and efficiency but, again, there is broad consensus on how to resolve these more detailed points.

The problems of finding ways to describe the *semantics* or meaning of programming languages proved to be far more challenging. A landmark conference was held in Baden bei Wien in 1964. This first ever IFIP working conference focussed on the subject of *Formal Language Description Languages* and most of the talks addressed proposals for ways to describe formally the semantics of programming languages. One article that had considerable influence on the work of the IBM Laboratory in Vienna was by the American John McCarthy. In [68],[6] he provides an *abstract interpreter* for *Micro-Algol*. (Essentially, an interpreter takes a program and a starting state and iteratively computes its final state; McCarthy used the adjective "abstract" to indicate that the metalanguage in which the interpreter was written was limited and mathematically tractable.) McCarthy's article provided an *operational semantics* for a very small language.

The influence of work at the IBM Vienna Lab on UK research becomes important below. The Vienna work in the 1960s saw the ideas of operational semantics extended and applied to the huge PL/I programming language. Their techniques became known as the ***Vienna Definition Language (VDL)***—see [63].

The task of formally describing the evolving PL/I language was undertaken separately from the language design team. Of course, there was strong interaction and communication. But, whenever the formalisation detected problems in the inherently ambiguous (and sometimes contradictory) natural language description, the formalists had to communicate the problem (sometimes by writing an indicative program). The response was then an amendment to the text, which again might not be crystal clear. The **lesson** here is that separation of designers from formalists is far less productive than working as an interdisciplinary team. This lesson is echoed below.

Turning to key UK speakers at the 1964 working conference, both Christopher Strachey and Peter Landin spoke about a more abstract approach than operational semantics: rather than define an abstract interpreter, they proposed that the constructs of a programming language should be translated into mathematical functions. Such functions were written in a notation known as the Lambda calculus. The development of what later became known as *denotational semantics*, is a fascinating story—suffice it here to say that researchers at Oxford University made the seminal contributions.[7] A key person in the Oxford-based research was the American Dana Scott.

At the beginning of the 1970s, researchers at the IBM Lab in Vienna also took the step from operational to denotational semantics. The PL/I language was again a catalyst: the Lab had been invited to build the PL/I compiler for a radically new machine architecture. Unfortunately, these machines were never built, but the aspect of VDM that related to describing language semantics was rescued by writing [13, 14]. A more detailed account of this work can be found in [6, 58] and the step from VDL to VDM is discussed in [56]. Here, again, there was a damaging wall between language designers[8] and the formalisers; moreover, the same mistake was repeated on a formal

[6] The proceedings of the conference [92] took some time to be published but are invaluable to those who want to understand the development of ideas because the post-presentation discussions were captured.

[7] An event was held in Oxford in November 2016 to mark the centenary of Strachey's birth and videos of the talks are available at: http://www.cs.ox.ac.uk/strachey100. An excellent biographical note on Strachey is [18].

[8] The source language of the compiler was to have been that of the ISO standard and the ISO PL/I standardisation committee were still evolving the language.

description of the machine architecture itself. In both cases, the separation proved wasteful and far less effective than if there had been a more tightly knit structure.

Returning to the Baden bei Wien conference, Tony Hoare expressed unease about all of the proposed techniques because he saw the need to leave some aspects of a language undefined. Obvious cases include features of a programming language that relate to implementations on specific machines. But Hoare's interjection was prescient because concurrent programs can legitimately yield different results depending on the rate of progress of their separate threads. This observation led Hoare to develop an *axiomatic approach* [46], which is key to reasoning about programs satisfying formal specifications.

The challenge of describing concurrent programming languages with their inherent nondeterminism was overcome by Gordon Plotkin's ***Structural Operational Semantics (SOS)*** [83].[9]

The history of formal semantic descriptions is covered at length in Troy Astarte's doctoral thesis [6] and more technical details are given in [58, 59]. (The formalisation of the SPARK-Ada language is discussed below in Section 4 below.)

3 CONSULTANT LED DEPLOYMENTS

One obvious mode of transferring ideas from academic originators into industrial practice is for the originators to act as consultants to practitioners. This offers a way of overcoming the inevitable lack of knowledge of novel ideas in the receiving organisation, but it runs several risks as outlined in the lessons spelled out at the end of this section. Specifically for formal methods ideas, this approach runs the risks associated with separation of practitioners from formalists mentioned in Section 2. Probably the best-known of these deployments was the work at IBM on their CICS system (description below), but there are also some useful lessons from a less well known activity in STL that is covered in Section 4.

The first issue is, of course, how to initiate the contact. Before coming to CICS itself, some background activities are worth outlining. The aspects of VDM associated with the semantic description of programming languages are mentioned in Section 2, but VDM is more widely known as a development method for (general) programs. Early work on the program specification and development aspects of VDM was actually undertaken inside IBM at the UK Laboratory in Hursley [53, 54]; the first book on this aspect of VDM was [55].

Starting in the 1970s, there was a programme of ***European Laboratories Integrated Professional Training Program*** (**ELIPT**) technical courses. A course based on the 1980 VDM book was offered and taught by Derek Andrews and Cliff Jones. Management at the IBM Laboratory in Germany at Böblingen made the decision to enrol most of their active programmers on this course and employees attended in more-or-less coherent project groups. A typical course would begin at a hotel in the *Schwarzwald* with two weeks of lectures and writing exercises followed by a one-week intensive workshop that initiated a formal description of a (simplified version of) the product of interest to that group of engineers.[10] Managers claimed that they were too busy to commit to this length of course and a shorter *Management Awareness* course had to be tailored to their needs. There were several significant success stories: One that is described in an external publication is [11].

[9] These lecture notes from 1981 were widely circulated and, thankfully, reprinted as [85]; they are accompanied by a useful commentary [84].

[10] Jones was involved in teaching of the first eight ELIPT-ASD courses; the majority took place in Germany, a couple in the United Kingdom and one in Italy.

In contrast to this organised enrolment, the IBM development laboratory at Hursley in the United Kingdom simply let individuals from any project enrol on the ELIPT course in a fairly random way. This meant that when needed (see below) there was no critical mass of engineers who were all up to speed on VDM. The **lesson** here is that education needs to establish a cohort of people in the receiving organisation.

To emphasise this lesson, it is worth comparing with Harlan Mills' success in IBM's ***Federal Systems Division*** (**FSD**) during the early 1970s.[11] Mills persuaded the director of FSD to attend the first course on his formal approach which ensured that no intermediate managers could claim they were too busy to take part. There was also a notion of "passing" the course. Effectively, most development engineers in FSD attended.

The story of using formal methods on CICS[12] began in 1980 and ran until 1993. During 1979–81, Jones was doing his (belated) doctorate under supervision of Tony Hoare at Oxford University. Jean-Raymond Abrial arrived at the ***Programming Research Group (PRG)*** at the same time as Jones. Hoare arranged that the two initially shared an office and many interesting discussions were conducted with Jones writing specifications in the VDM notation from [55] and Abrial experimenting with what was to become the Z notation (see below).

IBM's ***Customer Information Control System*** (**CICS**) is an on-line transaction processing system used by major financial and retail organisations. CICS had evolved from a customer's 1968 program to be a full-blown IBM product. In 1988, it consisted of well-over half a million lines of code written in at least two languages. This was the period of software "unbundling" and it was believed that CICS was at one time one of IBM's most profitable program products.

Because of his contacts with IBM Hursley[13] and his on-going courses, Jones was asked about ways to help the CICS team adopt formal specification. Starting in 1980, there were informal discussions that led, in early 1981, to the suggestion of a contract with a university (i.e., not with single consultants). Formal meetings between Hursley managers and Tony Hoare in the middle of 1981 resulted in a contract between IBM Hursley and Oxford University being in place by year end. Ib Sørensen and Tim Clement were to work on the project from the Oxford end; Hursley people included Pete Collins, John Wordsworth, and Peter Lupton; Hoare had asked that Jones worked as a consultant on the project[14] and Rod Burstall of Edinburgh University was involved in the same role.

As mentioned above, Abrial was developing the ideas that coalesced into the Z specification notation and it is clear that the challenges presented in describing CICS had an influence of this evolution. (It is often the case that there is an intellectual payback to proposers of methods when they face the challenges of applying ideas to problems larger than fit in academic articles.) Key discussion partners also included Bernard Sufrin and Carroll Morgan. Ian Hayes joined the project in January 1983, which was perhaps the key time for the development of Z's so-called "schema calculus". Hayes also went on to edit the first book on Z [42].[15] Other Z books from around this time include [69, 89, 90, 104] and Mike Spivey also programmed the first tool that type-checked Z specifications. A useful intermediate report on "CICS Experience with Z" is the (unrestricted) Technical Report [25]. The summary includes the following positive assessment:

[11] This is reported from Jones' memories of several personal discussions with Mills.
[12] See https://en.wikipedia.org/wiki/CICS.
[13] Jones had worked in Hursley 1965–68, 1970–73; the gaps filled by assignments to the IBM Lab Vienna and IBM's *European Systems Research Institute* in Belgium.
[14] Having submitted his thesis in June 1981, Jones moved to a chair in Manchester University starting August.
[15] The frustration at not having a stable reference document for Z in 1981 led to the construction of a spoof document with Sufrin's name shown as author but actually comprising a pastiche of other articles put together by researchers at PRG .

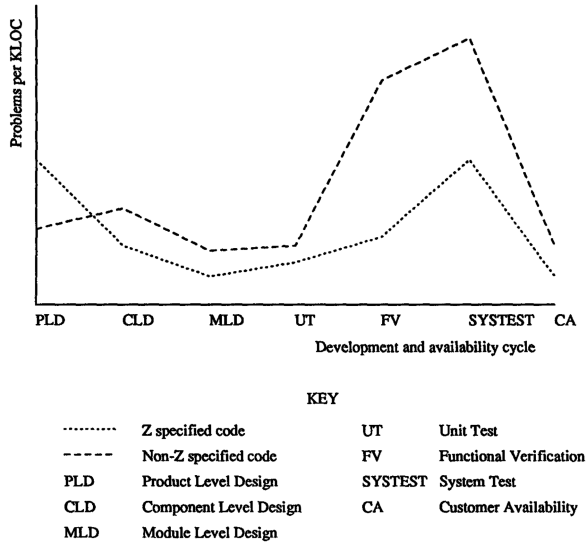

Fig. 1. Figure reproduced from [49] indicating errors found at different development stages. [Original caption] Comparison of problems found with two development methods for CICS/ESA V3.1.

> "From the industry point of view, this work has demonstrated that:
> — provided that there is adequate education and support, a mathematical notation such as Z can be used for software development in an industrial environment;
> — the use of formal methods changes the development process and brings greater precision to earlier stages;
> — communication between development groups can be improved."

Jim Woodcock worked on the Oxford end of the collaboration from 1985–1993. In particular, he sorted out the logic underlying Z [99]; he also co-authored a book on Z [100]. Steve King joined the project in January 1986 and stayed to the conclusion of the contract in 1993 (he spent a year working in Hursley [1990/1991]). The second edition of Hayes' *Case Studies* book [43, Part-IV] contains five chapters by Hayes and King on details of the description of parts of CICS.

A joint paper (Steve King—Oxford University's PRG and Iain Houston (IBM) [49] provides a strongly positive assessment of the exercise in formalisation. In particular, they address an oft-claimed benefit of formal methods, which is that more problems are detected early resulting in significantly reduced problems—and thus less costs—overall. The plausibility of the claim comes from the fact that having a model with precise meaning to both think about and discuss before implementation details are addressed results in clearer thinking than with pages of informal text possibly enhanced by diagrams with no firm meaning. (Similar experiences were seen when formal documents sharpened discussions in design reviews.) The figure reproduced here in Figure 1 provides strong evidence for the validity of this claim from a substantial industrial exercise.[16] The collaboration was recognised with a Queen's Award for Technological Achievement in 1992 to the Oxford PRG group and IBM Hursley.

Despite all of these positive indicators, an informal reunion of Hayes, Hoare, Jones, and Sørensen (kindly arranged by Jonathan Lawrence) in September 2011 found little trace of the continued use

[16]It should not surprise readers that IBM did not choose to publish numbers on the vertical scale.

of Z in IBM Hursley.[17] This prompts an evaluation of the **lessons** from what is widely viewed as one of the success stories of formal methods deployment.

- PRG staff who were funded by the collaborative contract had to spend significant amounts of time working on details of the CICS code;
- not only were general courses on Z designed and given to IBM engineers, but additional "readers' courses" were needed;
- the presence of "internal supporters" was crucial at the time of strongest interaction;
- there was a period when IBM engineers and management were pushing hard for a recognised standard for the Z notation;[18]
- (and linked to the previous point) there was a perceived need for tools that helped creation, maintenance and analysis of Z documents;
- management support is crucial (see above on Mills and IBM-FSD)—the attitude of IBM Hursley management ranged from supportive to antagonistic;[19]
- King and Jones ascribe the drift away from the use of Z in Hursley down to movements of people: there were probably not quite enough internal supporters, some moved on to other roles; less positive managers became responsible for decisions, which affected the selection of methods to be used in later releases.

An insider account by Ian Hayes and Steve King has been recently published [44].

Turning to another deployment, the software for the Sizewell-B nuclear reactor reinforces one of the key lessons. It was an extreme example of separation of development from verification and the responsible teams. Sizewell-B was the first nuclear reactor in the United Kingdom to have a programmable **primary protection system** (**PPS**). The software had been developed by the US company Westinghouse without a formal specification; there were about 100,000 lines of unique executable code.[20] Originally, there were two specification documents, a high level **Software Design Requirements** (**SDR**) and a more detailed **Software Design Specification** (**SDS**).

The regulator (Nuclear Installations Inspectorate) decided that it was necessary for the PPS software to be formally verified. The chosen route to (partial) formal verification was to manually translate (in the United Kingdom) the SDR and SDS into a mathematical specification and to write a translator from PL/M-86 to the **Intermediate Language** (**IL**) used as input to the MALPAS static analysis tool. This work started in January 1989 and was completed in 1993. The MALPAS analysis project team grew to more than 80 people and the project cost GBP 7 million. A review by Nuclear Electric [31] concluded that "The costs of the MALPAS review are high largely because the specification documents had to be manually translated into a formal notation before they could be used. This leads to the conclusion that the review processes need to be considered during the design phase of the project wherever possible".[21]

This clearly reinforces the lesson that a separation of developers and formalists is damaging in general and that leaving verification to the late stages of development is lengthy, difficult, and expensive.

An extremely important vector of formal methods research and deployment centres around Jean-Raymond Abrial; this story links to the United Kingdom (and involvement of the current

[17] Jonathan Lawrence drew Jones' attention to [48].

[18] An international standard for Z was approved in 2002.

[19] Jones' notes made at the time record one fairly senior manager responding to "it's just mathematical notation" with "I hate mathematics".

[20] It was written in PL/M-86 with some ASM86 and small amounts of PL/M-51 and ASM51.

[21] Ward [95] goes into detail on the MALPAS use of *Compliance Analysis* and explains that there was no choice but to develop pre and post condition specifications of components bottom up from the code.

authors) but is actually more international. After laying the groundwork of what became the Z notation, Abrial returned to France and acted as an independent consultant. He not only made a fundamental shift to create the B-method [1],[22] but he also developed a "B-tool" under contract to BP. Subsequently, Abrial developed a completely new "Atelier-B',' which was later supported by the French formal methods companies ClearSy and Systerel. The notation and tool were used in the important development of the software of Metro line 14 for RATP [27].

Abrial is a fascinating person who deserves a full biographical article. He is self-critical in the best possible way and has undertaken several complete rethinks of his ideas. His next step was strongly influenced by books on *Refinement Calculus* by Carroll Morgan [74], on one hand, and Ralph Back and Joakim von Wright [8], on the other. Abrial's *Event-B* is described in [3]. Tool support was designed and built in an EU-funded project known as *Rodin,* which was led by Newcastle University. The central tool activity was undertaken by Abrial, Laurent Voison, and Steffan Hallerstede at ETH Zurich in the chair of David Basin. A subsequent EU-funded project (*Deploy*)[23] was again led by Newcastle and this time involved four industrial companies. As its name suggests, the emphasis here was on deployment of *Event-B* and the *Rodin Tools*. The most accessible description is [86].

Although neither of the current authors were directly involved, it would be remiss in this section not to mention the work on hardware verification led by Mike Gordon of Cambridge—Larry Paulson wrote the Royal Society report [79].[24] Furthermore, the successful Oxford work on exploiting CSP by providing the FDR tool is described in an article by Bill Roscoe and Steve Brookes [16].

4 EXPERTISE WITHIN THE DEPLOYMENT ORGANISATION

This section addresses some organisations that made formal methods an integral part of their development process. Most if not all organisations that adopt formal methods will have progressed in stages of increasing rigour through informal and then structured methods, informed by their interactions with more advanced organisations and academics. One such organisation is the software engineering company Praxis (later Altran UK and more recently CapGemini UK) founded by one of the authors of this article (Thomas) with his colleague David Bean.

Thomas had worked on the design of a computer-based PABX at **Standard Telephones and Cables** (**STC**) in north London in 1975/6 where he was involved in introducing and teaching functional decomposition using the diagrammatic **Structured Analysis Design Technique** (**SADT**) [87]. Bean had worked in a leading UK software house, Logica, and was familiar with **Jackson Structured Programming** (**JSP**) [50]. They came together to set up the **South West Universities Regional Computer Centre** (**SWURCC**) where Thomas recruited a small team to develop an Algol68 compiler that was required by SWURCC's user community to run on the ICL 2980 mainframe computer.

The Algol68 compiler used a front end, Algol68RS [103] developed at the Royal Signals and **Radar Establishment** (**RSRE**) in Malvern, the United Kingdom by a team in the Mathematics Division that included Susan Bond, Ian Currie, John Morison, and Philip Woodward. The Mathematics Division at RSRE ran the establishment computing service, having developed the UK's first solid-state computer and written its operating system and compilers. RSRE also developed cryptographic systems, formal static analysis tools to help identify trojan code and provably secure hardware systems (the VIPER processor). The VIPER activity—for example [24]—is an interesting

[22] This book contains a generous acknowledgement to the influence of VDM on B.
[23] Other partners included SAP and Bosch—the Advisory Group was chaired by Thomas.
[24] An invaluable resource on theorem proving efforts is [65].

story in its own right[25] it used a formal hardware design and development system comprising the **Electronic Logic Language** (**ELLA**), developed by Morison and later released under a public license [75] supported by tools for design transformation, symbolic execution and formal verification. The SWURCC team impressed RSRE sufficiently that in addition to Algol68, they were commissioned to support and market ELLA.

SWURCC had built a reputation for software quality that led to them being invited to join a consortium (*Augusta*) funded by the UK Department of Trade and Industry to investigate the use of the new Ada programming language. Augusta was led by Tim Denvir of STC's telecommunications laboratory STL, who recalls that

> Our report, delivered in September 1981, took a few example problems, expressed a design following several different methods, and developed implementations from each in Ada. We also did a literature study of many more design methods and of developers. Among the mostly structured methods (such as JSD) we used and/or considered CCS and VDM [29].

The SWURCC team became the kernel of Praxis, set up by Thomas and Bean in 1983, transferring the ELLA and Algol68 compiler support and other projects from SWURCC including a Unix re-implementation for ICL and a FORTRAN compiler for a military version of ICL's Distributed Array Processor.

Denvir (the rest of this paragraph draws heavily on Denvir, op. cit.) explains that STL had significant history in the use of formal methods: by the mid 1970s they were already using Dijkstra's pre and post conditions to prove small programs correct. In January 1979, Denvir attended the Winter School on Abstract Software Specifications in Denmark with an STL colleague, Bernie Cohen, where they met Dines Bjørner, Cliff Jones, Steve Zilles, Joe Stoy, Peter Lucas, Peter Lauer, Barbara Liskov, Gordon Plotkin, Rod Burstall, David Park, O-J Dahl, Peter Mosses, and others. Denvir and Cohen were particularly impressed with lectures on VDM and in 1980 persuaded STC management to hire the services of Cliff Jones as a consultant to apply VDM to telecommunications projects. With sponsorship from Jack Shields, STC's Group Technical Director, STC developed their own courses on VDM and discrete mathematics. These were originally taught by Cliff Jones, then delivery was taken over by Tim Denvir, Roger Shaw, and Mel Jackson. Over 2–3 years they trained over 200 engineers from various group companies [52]. They experimented with Z [42] in one project using consultancy from Bernard Sufrin and Carroll Morgan from Oxford's PRG. With the support of the LFCS at Edinburgh University, hiring Mike Shields, Denvir's group developed an interest in the use of formal techniques for concurrent systems, and this culminated in a workshop held in Cambridge in September 1983 [28] and to the use of Robin Milner's Calculus of Communicating Sequential Processes (CCS) [72] in revising the standard for the telecommunications industry standard design language SDL.[26]

When the management changed at STC, Tim Denvir and Mel Jackson joined Praxis, bringing with them a European Commission contract to specify the ***Portable Common Tools Environment*** (**PCTE**) in an extended version of VDM (see [70, 71] for VVSL). Praxis was a fertile environment for the introduction of formal methods. They abandoned SADT and adopted VDM [55] and later Z [91] as specification and design methods, particularly for safety or security critical projects such as the protection system for a radiotherapy machine, the certification authority to support the MONDEX

[25] See [65, Chap 7] on VIPER; Donald Mackenzie's whole book contains a thorough and deep analysis of the concept and limitations of formal proofs about software.
[26] https://www.itu.int/rec/T-REC-Z.100/en.

smart card [39], a system (SHOLIS) to support the landing of helicopters on naval ships [62], and a system (CDIS) [37] to support the UK **National Air Traffic Services** (**NATS**) controllers handling aircraft for the five London airports.

Static dataflow and program analysis tools had been developed at RSRE based on research by Bob Philips of RSRE. The work was declassified in the 1970s so that it could be used on civilian safety critical projects, resulting in two commercially available products, the **Malvern Program Analysis Suite** (**MALPAS**) and the **Southampton Program Analysis Development Environment** (**SPADE**) (which became SPARK, see below) developed by a team led by Bernard Carré who were doing research in graph theory at Southampton University. Both MALPAS and SPARK have been used in the development and verification of a range of safety critical and security critical systems.

SPADE was originally developed to analyse and verify programs written in a small subset of Pascal but Ada was then chosen as the foundation for future work. An Ada subset (the SPADE Ada Kernel or SPARK) was defined informally by Bernard Carré and Trevor Jennings in 1987 and more formally defined using a variant of Z in *The Formal Semantics of SPARK* [66]. Bernard Carré set up a company **Program Validation Limited** (**PVL**) and later sold it to Praxis, with the PVL staff becoming Praxis employees. SPARK has kept up with revisions of Ada as each has been standardised and some industrial applications have been described by Chapman and Schanda [21]. A survey of Praxis/Altran projects has been published by White, Matthews, and Chapman [96]. The Critical Systems Division of Praxis was acquired by Altran UK in 1997.

It is important to stress that the Praxis' use of VDM, Z, CCS, and CSP was unusual if not unique. Most groups using formal methods used them to specify small, critical components; in Praxis these methods were used to specify system-level behaviour. This had a profound effect on the benefits and drawbacks that they found. These are some **lessons** from Praxis' experience with formal methods:

— Praxis developed its own courses to teach its software engineers to use VDM and Z. The courses lasted four days, preceded by a single day teaching discrete mathematics. This was found to be enough for computer science graduates to be able to read and start understanding a formal specification, though the ability to write good Z developed over a period of working on a project with access to experienced Z practitioners.
— Formal methods had to be fitted into an overall development process and combined with other techniques, for example, prototyping for user interfaces, and Dataflow Diagrams for process design. Furthermore, no single formal method covers every aspect so Praxis had to use different techniques for functionality and concurrency and find ways of integrating these. They also had to write specifications at the system level and at the design level and show that the design was consistent with the system specification. Using high-level, set-theory-based languages meant that there were almost no tools available and this limited their ability to generate executable prototypes, or carry out proofs, model checking, or automatic code generation.
— The benefits of this approach were a systematic, rigorous, and traceable development leading to systems with few defects in service [4, 39, 62, 93]; as Praxis improved their understanding of how to do this, they steadily drove down defect levels [38, 39].

Commercially, using formal methods had drawbacks as well as benefits:

— Customers were nervous about formal methods; the US *National Security Agency* asked Praxis to demonstrate the practicality of formal methods by developing an experimental system to control a secure enclave. The Tokeneer project was a success [10] in that the

NSA were unable to find any faults in the software, Praxis were able to train two NSA interns to extend the system, and the NSA said that the productivity of the Praxis team was the highest they had ever experienced but somehow this did not lead to sales of the SPARK toolset or to further NSA projects. The NSA later agreed that all of the Tokeneer specification, design, and development could be publicly released including all the tools and proofs,[27] so that anyone could download and study the project, experiment with the proof technology and see whether other tools might reveal defects that had not been found. The various follow-on projects and experiments have been summarised in a book chapter by Jim Woodcock [101].

— On the CDIS project for NATS, Praxis developed a formal VDM specification during the bid, to determine exactly what the customer meant by their requirements. This led to over 100 requests to NATS for clarifications, but it enabled Praxis to feel comfortable bidding for a large project whose cost exceeded Praxis' annual turnover[28] and to contract to repair at no charge any major faults that developed over the following five years. NATS later said that CDIS had been by far the easiest system to integrate that they had experienced. Furthermore, its stability was so good that, after a few years, NATS had to ask Praxis to retrain their staff in how to restart CDIS because it had failed so rarely that they were unsure how to do it.
— The functional specification for CDIS—and subsystem specifications for the two component parts (server and workstation)—were expressed in VVSL. The user interface used: state transition diagrams; for modelling concurrency, used CSP; and for the LAN design, used Milner's CCS. Praxis' technical architect, Anthony Hall, has described and explained the choice of methods [37]. With NATS' agreement, Praxis made the CDIS code and project records available for analysis by two academic researchers whose conclusions [82] were that formal methods can contribute to achieving very reliable code (but with many reservations, for which see the cited article).
— On CDIS, it was extremely useful to have a formal specification and traceable design documents available throughout the project for dealing with life cycle issues such as requirements changes (there were hundreds) and fault analysis. For a given requirements change request, it was possible to trace through from the formal specification to identify areas of the implementation affected by it and manage change in a cost-effective and efficient manner. With fault analysis, it was possible to trace back to the formal specification and identify whether the fault was due to an implementation or a requirements error. The use of a formal specification reduced costs and increased efficiency throughout the implementation life cycle.
— Formal methods projects of this sort need considerable work before any benefits are seen, which made it difficult to convince potential customers to invest in such projects.
— The lack of support tools can be a major problem for the use of formal methods that only address the specification stage. Some form of prototyping is very important so that clients can validate the specification and request changes while it is still relatively inexpensive to change it.
— System specifications need to be understood by domain experts as well as by computer scientists. Praxis rewrote the informal specifications using and including the formal specification so that it could be understood by clients. Preparing a formal specification facilitates writing a better structured, clearer, and shorter specification in natural language.

[27] https://www.adacore.com/tokeneer.
[28] Thomas remarked to Jones over a dinner in Brussels that he had "bet the company on VDM".

— Praxis offered some free warranties for projects that had a formal specification agreed with the client, as this made it possible to distinguish between errors and specification changes. This was less commercially risky than it might seem, because the warranty would be voided if the client changed the software themselves or asked another company to implement new features.
— NATS subsequently contracted Altran UK to develop a new ATC system iFACTS[29] that was successfully delivered using Z and SPARK. Thomas was present at one NATS meeting before this contract was awarded where the objection was raised that if NATS presented their regulator (The UK *Civil Aviation Authority*) with SPARK code and formal proofs as part of the evidence for the safety of iFACTS, there was a risk that the CAA would always require such strong evidence in future!

5 SOME INFLUENTIAL FACTORS

Apart from the lessons for those wishing to deploy formal methods, there are exogenous factors that have affected the growth and adoption of formal methods. This section covers some of these influences.

5.1 Research Funding

In the United Kingdom, funding from its research councils has been supportive to both the underlying research and deployment of formal methods. In fact, it can be argued that such funding was pivotal in the 1970s. It is also worth noting that BP's "Future ventures" programme provided funding for activities in Edinburgh University's *Laboratory for Foundations of Computer Science* (and for Edsger Dijkstra).

The body principally responsible for funding computer science research in theUnited Kingdom. Universities was then known as **Science Research Council** (**SRC**). The Computer Science committee, recognising the importance of distributed computing as a research area, appointed a panel in June 1976 under the chairmanship of Prof. I. Barron, to consider what action was necessary to encourage, coordinate, or direct research in Distributed Computing. The Distributed Computing Systems programme started in the academic year 1977–1978. DCS was the first attempt by SRC to establish a long term, extensive, and coordinated programme of research in Information Technology. The Technical Co-ordinators of DCS were Bob Hopgood [1977–1979], Rob Witty [1979–1981], and David Duce [1981–1984]. The primary scientific objectives of the programme were to seek an understanding of the principles of Distributed Computing Systems and to establish the engineering techniques necessary to implement such systems efficiently. These broad objectives reflect the relative immaturity of the subject when the programme was founded. In particular, the programme sought to establish an understanding of parallelism in information processing systems and to devise ways to take advantage of this. When the DCS programme was first established, the research covered five major topic areas, representing a progression from fundamental theory to novel applications. The areas were:

— Theory and Languages: An adequate theoretical basis for Distributed Computing Systems.
— Resource Management: Distribution of control, allocation, scheduling, and organization.
— (Machine) Architecture.
— Operational Attributes: Particularly reliability and performance.
— Design, Implementation, and Application: Hardware and software techniques for development and implementation.

[29] https://nats.aero/blog/2013/07/how-technology-is-transforming-air-traffic-management/.

A major theme in DCS was concerned with theories of parallel computation and with the development of notations and techniques for specifying and verifying such systems.[30]

The UK Alvey Programme ran from 1983–1987 and also invested in formal methods.[31] The focus of the Alvey programme [76] was pre-competitive advanced information technology research. It comprised four areas that seemed particularly relevant at the time:

— Software Engineering (led by David Talbot from ICL with Rob Witty from Informatics as his Deputy, who brought the focus on formal methods from DCS).
— Intelligent Knowledge-Based Systems.
— Man Machine Interaction.
— Advanced Microelectronics (VLSI Design).

Research was a collaboration between academia, government, and industry; it was directed into important areas and coordinated and the funding was substantial, GBP 350M at 1982 prices. The Programme put together 210 projects lasting, on average, three years and involving 2,500 people at its peak.

The largest Alvey project in *Software Engineering* funded Manchester University and ICL[32] to construct an integrated project support environment dubbed *IPSE 2.5* [88]. Researchers at Manchester and EPSRC's own Rutherford Lab delivered a theorem proving assistant *Mural* [60], which was licensed to *Winfrith Atomic Energy Establishment*.

Another activity under the Alvey programme led to the creation of a handbook of formal methods. The contents attempted to identify areas of applicability for notations such as VDM, Z, CSP, and CCS.

Brian Oakley led the UK Alvey programme and wrote:

> . . . the main achievement of the Alvey Software Engineering Programme is the success with which 'Formal Methods' from the academic world have been pulled through to industrial use. The implications of this achievement are difficult to overestimate, for these Formal Methods are the route to much better software writing, and the economic consequences will be considerable—on a par with those of the revolution in civil engineering in the last century.

As recently as the 2010s, formal methods were still identified as an area for growth of EPSRC funding.

Funding from the various European Union research framework programmes has also been a significant aid to formal methods research and deployment. Again focussing on items where the authors have first-hand knowledge, one of the longest-lasting impacts started with the funding of an activity called *VDM-Europe*: meetings of experts in Brussels were supported for several years and led to the first symposium of *VDM-Europe* [15]. These conferences morphed[33] into *FM-E*,[34] which not only organises a highly-rated symposium at roughly 18-monthly intervals, but has also

[30]See http://www.chilton-computing.org.uk/acd/dcs/overview.htm and http://www.chilton-computing.org.uk/inf/literature/reports/alvey_report/overview.htm.
[31]See for example Alvey News SE2/18 that contains a list of some of the projects funded by the Software Engineering Programme.
[32]This project came close to non-submission when, having crafted a neat collaboration of three industrial organisations (STL, IDEC, and ICL), Jones was informed that they were merging and that the combination of their individual intended commitments was not defensible to a single board of directors. Brian Warboys then of ICL steered a tense period of revisions at the eleventh hour.
[33]Jim Woodcock tackled Jones about widening the remit of VDM-Europe to include other specification languages.
[34]see http://www.fmeurope.org/.

held three World Congresses [19, 67, 97] and has widened its venues to North America, Singapore (and the 2021 event is planned for China).

5.2 Tool Support of Formalism

The question of how much the adoption of formal methods is influenced by the availability of tool support is interesting but is not universally agreed. Early on, large formal descriptions were constructed with minimal tool support. A significant example is the formal description of PL/I from the IBM Vienna Lab. It is probably fair to say that the risks of introducing inconsistencies are far higher when a document is revised than when it is first constructed. It is, however, clearly short-sighted not to at least syntax and type check any large block of formulae.

The question is how much further one can go without the tool support becoming an end in itself and possibly even distracting from the thought process that is crucial to the construction of an abstract model. Jim Horning (private communication) captured one of the reservations about tools with his phrase "mental versus metal tools". At least some of the differences in people's evaluation of the role of tools can be accounted for by the contribution they hope to result from using formalism.

It is relatively easy to persuade organisations to use tools that analyse finished code in order to detect potential errors. There are sub-issues here: Peter O'Hearn (see Section 6.2) points out the tools that detect too many false positives are unlikely to endear themselves to developers. But, broadly, using model-checking tools in a development process that does not require deep understanding of formal methods from the developers is an easier sell than starting out by insisting that developers must employ formalism in the specification and early design phases.

The authors of the current article; however, both argue that the real payoff of formal methods comes from their use early in the design phase. Thomas has written [94] about the cost-effectiveness of using formal methods early; the *Tokeneer* study mentioned in Section 4 supports this view; Figure 1 provides evidence that formalism used to front-load thinking pays off; a similar result can be seen in the dual-track study reported in [17].

Many companies are willing to buy tools and see them as a quick fix—but fail to recognise that tools exist to support methods and the main investment has to be in adopting the methods.

The view of "mental tools" in no way removes the need for tool support but it does moderate the extent to which the tool should be allowed to become the master of the method. For example, a large formal text might be regarded as an obvious input to a theorem proving system. Unfortunately, there are few success stories of such efforts.[35] The reason would appear to be that theorem proving systems not only require learning another formalism but that their modes of interaction distract from thinking about the application in hand. There are numerous stories of formal machine-checked proofs that do not actually capture what the user intended to establish.

Examples of tools that offer fairly direct support for established specification languages include a VDM tool from Adelard[36] and the IFAD Toolset (also for VDM) that has subsequently been developed into an open-source Overture tool. Probably the most significant set of tools comes from Abrial, with the Rodin tool support[37] for Event-B being the most recent.

5.3 Standards

One way in which the use of formalism could be encouraged is via standards. In May 1989, the UK **Ministry of Defence (MoD)** issued a draft Interim Defence Standard 00-55 "Requirements for

[35] The US work starting with the Boyer–Moore prover through to ACL/2 is a notable achievement—see [73].
[36] The work of the Adelard company would justify an article of its own. For example, their development of the "Dust Expert" software is reported in [23].
[37] See http://www.event-b.org/.

the procurement of safety critical software in defence equipment" for comment. The draft standard required that safety critical software should be formally specified and formally verified. Several companies in the defence industry attempted to persuade the MoD to withdraw the draft standard; but the MoD issued it as an interim standard in 1991 and made the standard mandatory for the SHOLIS project [62]. The interim standard was replaced by a full version in 1997,[38] retaining the requirement for formal methods and including several examples from the SHOLIS project. Def Stan 00-55 issue three recommended the use of civil standards such as RTSA DO-178, ISO 61508, and RTSA DO-254.

IEC 61508 is the international standard for functional safety of programmable electronic systems. It requires that each safety function has a Safety Integrity Level that defines the allowable probability of failure: for continuous control of the most safety critical function (SIL4) the allowable probability must not exceed 10^{-8}/hour. The standard recommends the use of formal methods for a SIL4 software safety function but does not mandate their use. In successive revisions of the standard, major European companies have repeatedly frustrated attempts to make formal methods mandatory for SIL 4 software.

Returning to Harlan Mills and what he achieved in IBM's Federal Systems Division (and again relying on Jones' memory of personal discussions with Mills) perhaps he had the best approach to standards. At one point in time, an FSD standard for software developers stated that programmers should accompany loops with an indication of why they were claimed to achieve their aim; there was not a mandated style for such annotations but the document did offer an example of a style that would serve.

There is of course also the question of standards for the formalism itself. It was mentioned in Section 3 that, during the CICS effort, there was a request from IBM for a standard for the Z notation itself. This is an understandable wish in that it opens up the possibility of sourcing tools and expertise from different organisations. In fact in 1996, VDM was earlier in obtaining an ISO standard[39] and the Z standard followed in 2002.[40]

6 CONCLUSIONS

Our emphasis in this article has been on lessons that can be derived from early attempts to apply research on formal methods in significant software development. This should in no way be seen as expressing reservations about the potential of the ideas and Section 6.2 points to two important recent success stories. If we cannot learn from earlier difficulties, no progress is made and it is hoped that the lessons might help other researchers or commercial organisations, which are planning to experiment with formal methods. Rather than re-list all of the lessons noted earlier in the article, Section 6.1 pinpoints a few key messages. It should also be mentioned that this article was written on the expectation that it would accompany ones that relate to formalism in other European countries and that this is the reason for the UK-centric account in the current article.

6.1 Lessons

The contributions of UK researchers to the fundamental ideas that have shown how formal concepts and notations can be used in the description and development of software are significant. Rather than list and attribute the scientific source material, we have in this article identified some examples of attempts to deploy the theory into practical environments. As indicated at the beginning of the article, we have mainly reported on deployments of which we have first hand

[38] http://www.software-supportability.org/Docs/00-55_Part_1.pdf.
[39] See https://www.iso.org/standard/22988.html.
[40] See https://www.iso.org/standard/21573.html.

knowledge. These close encounters have made it possible to analyse the difficulties that were experienced.

Probably the most important single difficulty that complicated early deployments was the relatively small number of people available in receiving organisations who had acquaintance with a broad knowledge of theoretical concepts. A short course on one or another specific notation does not fully equip someone to apply that notation to the key aim of abstracting away from the details of potential implementations; similarly, nor does being shown a few examples of proofs convey the fundamental idea of recording a convincing correctness argument. The more recent experiences from major companies like Microsoft, Amazon, and Facebook suggest that the general educational environment now provides a far better basis than was available last century.

The attitude and commitment of management is clearly related and also of major importance. The graph in Figure 1 indicates a challenge for managers who only feel comfortable when they can "weigh the code": just as in all engineering endeavours, care, and thought early in development clearly pays off later but does not yield immediate lines of code. (Nor does the drawing of careful architectural plans lay any bricks.)

Linking the points about technical expertise and management commitment is the issue of whether the key expertise is inside the receiving organisation or supplied by external consultants. Key advantages that come from the research expertise being within the deployment organisation are bandwidth of communication and stability. Perhaps more important is the avoidance of a split between "practical" work and *post hoc* formalisation. This division appears to have been a source of problems in many of the early attempts to gain benefit from using formal methods in industry.

Another issue is the choice of project—perhaps crucially the first project. A significant success story that is located about as far from our stated geographic focus as can be is the work in Australia at *CSIRO Data61* (previously known as NICTA). Gerwin Klein and his team recognised the importance of microkernels because weaknesses here open any software built on top of them to subversion. They developed the microkernel called *sel4,* which is described in [45]—earlier publications can be traced from its references. In today's world where almost everything depends on software, it is sometimes a financial aspect that identifies a development as "business critical". There is, of course, the class of "safety critical" systems that comprised early deployments of formal methods. The *sel4* exercise is a reminder that underlying software can provide a Trojan Horse on which reliance should only be proportional to its demonstrated trustworthiness.

One closing lesson (and perhaps an uncomfortable one for the current authors) was mentioned by Jonathan Lawrence when he kindly reviewed the material relating to the IBM-CICS project: he suggested that one should "not try to be too ambitious". It would be legitimate to ask whether some of the early deployments suffered precisely because researchers wanted to see the full extent of their research put into practice even if the receiving organisation was unready.

6.2 More Recent Work

Much could be written about more recent attempts to use formal methods in practical engineering contexts and we hope that the current article will prompt authors who have first-hand experience of such projects to contribute articles, which also draw out general lessons (it is tempting to trumpet successes, but it can offer more help to the community to also record difficulties).

There is considerable room for optimism in the use of formal methods in companies whose business is software (this positive view of the future is reflected in the survey contained in [32][41]).

[41] Which it is interesting to compare to the historical [22].

Recent such attempts include those spearheaded by Peter O'Hearn at Facebook and Byron Cook at ***Amazon Web Services*** (**AWS**). O'Hearn made major contributions to *Concurrent Separation Logic* (e.g., [78]) and went on to form, with colleagues, *Monoidics* which was acquired by Facebook in 2013. The group has worked inside Facebook and [77] reports considerable success in creating tools (see [30]) that are used in the standard development cycle by Facebook engineers.[42] In a private conversation with Jones, O'Hearn attributed the positive adoption by practicing engineers both to the creation of apposite tools and the fact that the general knowledge of fundamental computer science ideas is much more widespread now than in the attempts reported on earlier in this article that mainly date from the last century. Two additional points need making. One is that a huge part of the impact of O'Hearn and his colleagues is that they made the decision to work inside the industrial engineering groups (see lessons above); messages such as the unacceptability of rafts of false positive diagnostics are much louder when working alongside practicing engineers. The second point is a concern that some universities appear to be softening their coverage of formal material in order to make computing a more popular subject.

Byron Cook's [26] is one of a sequence of articles reporting on the application of formal methods at AWS; his recorded keynote[43] talk at FLoC-18 in Oxford is inspirational and, related to the lessons about management commitment, even more telling are the talks from senior managers at AWS.[44]

It is also important to note that many of the methods described earlier in the current article continue to thrive: interested readers could consult the sites of companies such as Clearsy (https://www.clearsy.com/en/) or Systerel (https://www.systerel.fr/) as well as those cited above. Some methods have evolved including VDM's support of object-oriented notation (see https://www.overturetool.org/languages/). There have also been successes in combining process algebraic notations with state-based specifications; to point to just one example Circus (https://www.cs.york.ac.uk/circus/) has good tool support.

The relentless growth of new areas of computing presents fresh challenges for formal methods: see for example [20, 64] on Robotics. Finally, it would be remiss not to point to a conference held in 2019 that has a good collection of articles [7] on historical material related to formalisms.

ACKNOWLEDGMENTS

A talk to a meeting of the PROGRAMme project (ANR-17-CE38-0003-01) triggered the first draft of the current article. Troy Astarte offered valuable comments on an early draft of the article.

The authors are indebted for input on specific aspects of this article:

— IBM/CICS: from Tim Clement, Ian Hayes, Steve King, Jonathan Lawrence, and Carroll Morgan;
— STL: from Tim Denvir and Mel Jackson;
— Praxis and Altran UK: from David Bean, Roderick Chapman, Anthony Hall, and Martyn Ould;
— UK Research Councils: from Mel Jackson, David Duce, and Rob Witty.

The authors are grateful for the useful and detailed comments from the anonymous reviewers. All remaining errors and opinions are, however, the responsibility of the authors.

REFERENCES

[1] J.-R. Abrial. 1996. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press.

[42] The impact is particularly interesting in an environment that is described as "move fast and break things" [40].
[43] https://www.youtube.com/watch?v=JfjLKBO27nw.
[44] https://www.youtube.com/watch?v=x6wsTFnU3eY; https://www.youtube.com/watch?v=BbXK_-b3DTk.

[2] J.-R. Abrial. 2010. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY.
[3] J.-R. Abrial. 2010. *The Event-B Book*. Cambridge University Press, Cambridge, UK.
[4] Peter Amey. 2002. Correctness by construction: Better can also be cheaper. *CrossTalk: the Journal of Defense Software Engineering* 2 (3 2002), 24–28.
[5] Krzysztof R. Apt and Ernst-Rüdiger Olderog. 2019. Fifty years of Hoare's logic. *Formal Aspects of Computing* 31, 6 (2019), 751–807.
[6] Troy K. Astarte. 2019. *Formalising Meaning: a History of Programming Language Semantics*. Ph.D. Dissertation. Newcastle University.
[7] Troy K. Astarte (Ed.). 2020. *HFM 2019 - History of Formal Methods Workshop. In: Formal Methods. FM 2019 International Workshops. Porto, Portugal, October 7–11, 2019, Revised Selected Papers, Part II*. Number 12233 in Lecture Notes in Computer Science. Springer-Verlag. DOI: https://doi.org/10.1007/978-3-030-54997-8
[8] R. -J. R. Back and J. von Wright. 1998. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, New York.
[9] John W. Backus, Friedrich L. Bauer, Julien Green, Charles Katz, John McCarthy, Peter Naur, Alan J. Perlis, Heinz Rutishauser, Klaus Samelson, Bernard Vauquois, et al. 1960. Report on the algorithmic language ALGOL 60. *Numerische Mathematik* 2, 1 (1960), 106–136.
[10] J. Barnes, R. Johnson, J. C. Widmaier, B. Everett, R. Chapman, and Cooper D. 2006. Engineering the Tokeneer enclave protection software. In *Proceedings of IEEE International Symposium on Secure Software Engineering*, S. Redwine, Hall A., and J. Wing (Eds.). IEEE Computer Society.
[11] F. Beichter, Otthein Herzog, and Heiko Petzsch. 1983. SLAN-4: A language for the specification and design of large software systems. *IBM Journal of Research and Development* 27, 6 (1983), 558–576.
[12] D. Besnard, C. Gacek, and C. B. Jones (Eds.). 2006. *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*. Springer-Verlag. DOI: https://doi.org/10.1007/b138838
[13] D. Bjørner and C. B. Jones (Eds.). 1978. *The Vienna Development Method: The Meta-Language*. Lecture Notes in Computer Science, Vol. 61. Springer-Verlag. DOI: https://doi.org/10.1007/3-540-08766-4
[14] Dines Bjørner and Cliff B. Jones (Eds.). 1982. *Formal Specification and Software Development*. Prentice Hall International. Retrieved from http://homepages.cs.ncl.ac.uk/cliff.jones/ftp-stuff/BjornerJones1982.
[15] Dines Bjørner, C. B. Jones, M. Mac an Airchinnigh, and E. J. Neuhold (Eds.). 1987. *VDM – A Formal Definition at Work*. Lecture Notes in Computer Science, Vol. 252. Springer-Verlag. DOI: https://doi.org/10.1007/3-540-17654-3
[16] Steve Brookes and Bill Roscoe. 2021. CSP: A practical process algebra. See [61], Chapter 9. DOI: https://doi.org/10.1145/3477355
[17] T. M. Brookes, John S. Fitzgerald, and Peter Gorm Larsen. 1996. Formal and informal specifications of a secure system component: Final results in a comparative study. In *Proceedings of the FME'96: International Symposium of Formal Methods Europe (Lecture Notes in Computer Science)*, Vol. 1051. Springer-Verlag, 214–227.
[18] Martin Campbell-Kelly. 1985. Christopher Strachey, 1916-1975: A biographical note. *IEEE Annals of the History of Computing* 1, 7 (1985), 19–42.
[19] Ana Cavalcanti and Dennis Dams (Eds.). 2009. *FM 2009: Formal Methods: Second World Congress, Eindhoven, The Netherlands, November 2–6, 2009, Proceedings*. Vol. 5850. Springer.
[20] Ana Cavalcanti, Brijesh Dongol, Rob Hierons, Jon Timmis, and Jim Woodcock (Eds.). 2021. *Software Engineering for Robotics*. Springer.
[21] R. Chapman and F. Schanda. 2014. Are we there yet? 20 years of industrial theorem proving with SPARK. In *Proceedings of the Interactive Theorem Proving (Lecture Notes in Computer Science)*, G. Klein and R. Gamboa (Eds.), Vol. 8558. Springer-Verlag, 17–26.
[22] Edmund M. Clarke, Jeannette M. Wing, et al. 1996. Formal methods: State of the art and future directions. *ACM, Computing Surveys* 28, 4 (12 1996), 626–643. DOI: https://doi.org/10.1145/242223.242257
[23] Tim Clement, Ian Cottam, Peter Froome, and Claire Jones. 1999. The development of a commercial "shrink-wrapped application" to safety integrity level 2: The DUST-EXPERT™ story. In *Proceedings of the International Conference on Computer Safety, Reliability, and Security*. Springer-Verlag, 216–225.
[24] A. Cohn. 1988. A proof of correctness of the VIPER microprocessor: The first level. In *Proceedings of the VLSI Specification, Verification and Synthesis*, G. Birtwistle and P.A. Subrahmanyam (Eds.). Vol. 35. Kluwer.
[25] B. P. Collins, J. E. Nicholls, and I. H. Sorensen. 1987. *Introducing Formal Methods: The CICS Experience with Z*. Technical Report TR12.060. IBM Hursley Laboratory.
[26] Byron Cook. 2018. Formal reasoning about the security of Amazon Web Services. In *Proceedings of the Computer Aided Verification. CAV 2018 (Lecture Notes in Computer Science)*, Weissenbacher G. Chockler H. (Ed.), Vol. 10981. Springer-Verlag, 38–47.
[27] Babak Dehbonei and Fernando Mejia. 1994. Formal methods in the railways signalling industry. In *Proceedings of the International Symposium of Formal Methods Europe (Lecture Notes in Computer Science)*, Vol. 873. Springer-Verlag, 26–34.

[28] B. T. Denvir, W. T. Harwood, M. I. Jackson, and M. J. Wray. 1985. *The Analysis of Concurrent Systems: Cambridge, September 1983, Proceedings of a Workshop*. Lecture Notes in Computer Science, Vol. 207. Springer Verlag, Berlin.
[29] T. Denvir. 2017. Fifty years of formal methods in software engineering. *FACTS* (8 2017), 16–26. Retrieved from https://cdn.bcs.org/bcs-org-media/3080/facs-aug17.pdf.
[30] Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling static analyses at Facebook. *Communications of the ACM* 62, 8 (2019), 62–70.
[31] T. Fenney. 1995. *Sizewell B Primary Protection System. An Assessment of the Confirmatory Review Processes*. Technical Report. Nuclear Electric plc.
[32] Hubert Garavel, Maurice H. ter Beek, and Jaco van de Pol. 2020. The 2020 expert survey on formal methods. In *Proceedings of the Formal Methods for Industrial Critical Systems*, Maurice H. ter Beek and Dejan Ničković (Eds.). Springer, 3–69.
[33] Mario Gleirscher, Simon Foster, and Jim Woodcock. 2019. New opportunities for integrated formal methods. *ACM Computing Surveys* 52, 6 (Oct. 2019), 1–36. DOI: https://doi.org/10.1145/3357231
[34] Mario Gleirscher and Diego Marmsoler. 2020. Formal methods in dependable systems engineering: A survey of professionals from Europe and North America. *Empirical Software Engineering* 25, 6 (2020), 4473–4546. DOI: https://doi.org/10.1007/s10664-020-09836-5
[35] Mike Gordon. 2000. From LCF to HOL: A short history. In *Proceedings of the Proof, Language, and Interaction*. 169–186.
[36] M. Gordon, R. Milner, and C. Wadsworth. 1979. *Edinburgh LCF*. Lecture Notes in Computer Science, Vol. 78. Springer-Verlag. DOI: https://doi.org/10.1007/3-540-09724-4
[37] Anthony Hall. 1996. Using formal methods to develop an ATC information system. *IEEE Software* 13, 2 (1996), 66–76. Hard copy.
[38] Anthony Hall. 2005. Realising the benefits of formal methods. In *Proceedings of the Formal Methods and Software Engineering (Lecture Notes in Computer Science)*, Vol. 3785. Springer-Verlag, 1–4.
[39] Anthony Hall and Roderick Chapman. 2002. Correctness by construction: Developing a commercial secure system. *IEEE Software* 19, 1 (2002), 18–25.
[40] Mark Harman and Peter O'Hearn. 2018. From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis. In *Proceedings of the 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation* . IEEE, 1–23.
[41] Ian Hayes (Ed.). 1993. *Specification Case Studies* (second ed.). Prentice Hall International, Englewood Cliffs, N.J.
[42] I. J. Hayes (Ed.). 1987. *Specification Case Studies*. Prentice Hall International.
[43] I. J. Hayes (Ed.). 1992. *Specification Case Studies* (second ed.). Prentice Hall International. 332 pages.
[44] Ian J. Hayes and Steve King. 2021. Software specification. See [61], Chapter 11. DOI: https://doi.org/10.1145/3477355
[45] Gernot Heiser, Gerwin Klein, and June Andronick. 2020. seL4 in Australia: From research to real-world trustworthy systems. *Communications of the ACM* 63, 4 (2020), 72–75.
[46] C. A. R. Hoare. 1969. An axiomatic basis for computer programming. *Communications of the ACM* 12, 10 (1969), 576–580.
[47] C. A. R. Hoare. 1985. *Communicating Sequential Processes*. Prentice Hall.
[48] Jonathan Hoare, Jeremy Dick, Dave Neilson, and Ib Sørensen. 1996. Applying the B technologies to CICS. In *Proceedings of the FME'96: Industrial Benefit and Advances in Formal Methods (Lecture Notes in Computer Science)*, M.-C. Gaudel and J. Woodcock (Eds.), Vol. 1051. Springer-Verlag, 74–84.
[49] Iain Houston and Steve King. 1991. CICS project report experiences and results from the use of Z in IBM. In *Proceedings of the VDM'91 Formal Software Development Methods (Lecture Notes in Computer Science)*, S. Prehn and W. J. Toetenel (Eds.), Vol. 551. Springer-Verlag, 588–596.
[50] Michael Jackson. 1975. *Principles of Program Design*. Academic Press.
[51] Michael Jackson. 2000. *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley.
[52] M. I. Jackson, B. T. Denvir, and R. C. Shaw. 1985. Experience of introducing the vienna development method into an industrial organisation. In *Proceedings of the Formal Methods and Software Development. TAPSOFT 1985 (Lecture Notes in Computer Science)*, H. Ehrig, C. Floyd, M. Nivat, and J. Thatcher (Eds.). Springer-Verlag.
[53] C. B. Jones. 1971. *Development of Correct Programs: An Example Based on Earley's Recogniser*. Technical Report TN 9000. IBM Laboratory, Hursley.
[54] C. B. Jones. 1973. *Formal Development of Programs*. Technical Report 12.117. IBM Laboratory Hursley.
[55] C. B. Jones. 1980. *Software Development: A Rigorous Approach*. Prentice Hall International, Englewood Cliffs, N.J. Retrieved from http://portal.acm.org/citation.cfm?id=539771.
[56] C. B. Jones. 2001. The transition from VDL to VDM. *Journal of Universal Computer Science* 7, 8 (2001), 631–640. DOI: https://doi.org/10.3217/jucs-007-08-0631
[57] Cliff B. Jones. 2003. The early search for tractable ways of reasoning about programs. *IEEE Annals of the History of Computing* 25, 2 (2003), 26–49. DOI: https://doi.org/10.1109/MAHC.2003.1203057

[58] Cliff B. Jones and Troy K. Astarte. 2016. *An Exegesis of Four Formal Descriptions of ALGOL 60*. Technical Report CS-TR-1498. Newcastle University School of Computer Science.
[59] Cliff B. Jones and Troy K. Astarte. 2018. Challenges for semantic description: Comparing responses from the main approaches. In *Proceedings of the 3rd School on Engineering Trustworthy Software Systems (Lecture Notes in Computer Science)*, Jonathan P. Bowen, Zili Zhang, and Zhiming Liu (Eds.), Vol. 11174. 176–217. DOI: https://doi.org/10.1007/978-3-030-02928-9_6
[60] C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore. 1991. *Mural: A Formal Development Support System*. Springer-Verlag. Retrieved from http://homepages.cs.ncl.ac.uk/cliff.jones/ftp-stuff/mural.pdf.
[61] Cliff B. Jones and Jayadev Misra (Eds.). 2021. *Theories of programming: The Life and Works of Tony Hoare*. ACM. DOI: https://doi.org/10.1145/3477355
[62] Steve King, Jonathan Hammond, Rod Chapman, and Andy Pryor. 2000. Is proof more cost-effective than testing? *IEEE Transactions on Software Engineering* 26, 8 (2000), 675–686.
[63] Peter Lucas and Kurt Walk. 1969. On the formal description of PL/I. *Annual Review in Automatic Programming* 6 (1969), 105–182.
[64] Matt Luckcuck, Marie Farrell, Louise A. Dennis, Clare Dixon, and Michael Fisher. 2019. Formal specification and verification of autonomous robotic systems: A survey. *Computing Surveys* 52, 5 (2019), 100.
[65] Donald MacKenzie. 2001. *Mechanizing Proof: Computing, Risk, and Trust*. MIT Press.
[66] Marsh and O'Neil. 1994. The Formal Semantics of SPARK. Praxis UK, Bath, England
[67] ter Beek Maurice H. Annabelle McIver, and José Oliveira (Eds.). 2019. *Formal Methods – The Next 30 Years*. Lecture Notes in Computer Science, Vol. 11800. Springer-Verlag.
[68] John McCarthy. 1966. A formal description of a subset of ALGOL. In *Proceedings of the Formal Language Description Languages for Computer Programming*. North-Holland, 1–12.
[69] Mike McMorran and Steve Powell. 1993. *Z Guide for Beginners*. Alfred Waller Limited.
[70] Cornelis Adam Middelburg. 1989. VVSL: A language for structured VDM specifications. *Formal Aspects of Computing* 1, 1 (1989), 115–135.
[71] C. A. Middelburg. 1990. *Syntax and Semantics of VVSL: A Language for Structured VDM Specifications*. Ph.D. Dissertation. PTT Research, Leidschendam, Department of Applied Computer Science.
[72] R. Milner. 1980. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science, Vol. 92. Springer-Verlag. DOI: https://doi.org/10.1007/3-540-10235-3
[73] J. Strother Moore. 2019. Milestones from the Pure Lisp theorem prover to ACL2. *Formal Aspects of Computing* 31, 6 (2019), 699–732.
[74] C. C. Morgan. 1994. *Programming from Specifications* (second ed.). Prentice Hall.
[75] J. D. Morison and A. S. Clarke. 1993. *Ella 2000: A Language for Electronic System Design*. McGraw Hill.
[76] Brian Oakley and Kenneth Owen. 1990. *Alvey: Britain's Strategic Computing Initiative*. MIT press.
[77] Peter O'Hearn. 2015. From categorical logic to Facebook engineering. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, 17–20.
[78] P. W. O'Hearn. 2007. Resources, concurrency and local reasoning. *Theoretical Computer Science* 375, 1–3 (5 2007), 271–307.
[79] Lawrence C. Paulson. 2018. Michael john caldwell gordon (FRS 1994), 28 February 1948–22 August 2017. (2018). Royal Society obituary.
[80] Lawrence C. Paulson, Tobias Nipkow, and Makarius Wenzel. 2019. From LCF to Isabelle/HOL. *Formal Aspects of Computing* 31, 6 (2019), 675–698.
[81] Charles Sanders Peirce. 1991. *Peirce on Signs: Writings on Semiotic*. UNC Press Books.
[82] S. L. Pfleeger and L. Hatton. 1997. Investigating the influence of formal methods. *Computer* 30, 2 (2 1997), 33–43.
[83] G. D. Plotkin. 1981. *A Structural Approach to Operational Semantics*. Technical Report DAIMI FN-19. Aarhus University.
[84] Gordon D. Plotkin. 2004. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming* 60–61, 1 (July–December 2004), 3–15. DOI: https://doi.org/doi:10.1016/j.jlap.2004.03.009
[85] Gordon D. Plotkin. 2004. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming* 60–61 (July–December 2004), 17–139. DOI: https://doi.org/doi:10.1016/j.jlap.2004.03.002
[86] Alexander Romanovsky and Martyn Thomas. 2013. *Industrial Deployment of System Engineering Methods*. Springer-Verlag.
[87] D. T. Ross. 1977. Structured analysis (SA): A language for communicating ideas. *IEEE Transactions on Software Engineering* SE-3, 1 (1977), 16–34.
[88] R. A. Snowdon. 1990. An introduction to the IPSE 2.5 project. In *Proceedings of the Software Engineering Environments (Lecture Notes in Computer Science)*, F. Long (Ed.), Vol. 467. Springer-Verlag.
[89] J. M. Spivey. 1992. *The Z Notation: A Reference Manual* (second edition ed.). Prentice Hall International.

[90] J. M. Spivey. 1988. *Understanding Z—A Specification Language and its Formal Semantics*. Number 3 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.
[91] J. M. Spivey. 1989. *The Z Notation: A Reference Manual*. Prentice Hall International.
[92] T. B. Steel (Ed.). 1966. *Formal Language Description Languages for Computer Programming*. North-Holland.
[93] Praxis Critical Systems. 2006. Spark: A successful contribution to the Lockheed C130-J Hercules II. Praxis report.
[94] Martyn Thomas. 1993. The industrial use of formal methods. *Microprocessors and Microsystems* 17, 1 (1993), 31–36.
[95] N. J. Ward. 1993. The rigorous retrospective static analysis of the Sizewell 'B' Primary Protection System software. In *Proceedings of the International Conference on Computer Safety, Reliability and Security SAFECOMP'93*, Górski J. (Ed.). Springer-Verlag, 171–181.
[96] N. White, S. Matthews, and R. Chapman. 2017. Formal verification: Will the seedling ever flower? *Phil Trans R Soc A* 375, 2104 (2017), 1–14. DOI: https://doi.org/10.1098/rsta.2015.0402
[97] J. M. Wing, J. Woodcock, and J. Davies (Eds.). 1999. *Formal Methods FM'99*. Lecture Notes in Computer Science, Vol. 1709. Springer-Verlag.
[98] Niklaus Wirth. 1977. What can we do about the unnecessary diversity of notation for syntactic definitions? *Communications of the ACM* 20, 11 (1977), 822–823. DOI: https://doi.org/10.1145/359863.359883
[99] J. C. P. Woodcock and S. M. Brien. 1992. W: A logic for Z. In *Proceedings of the Z User Workshop, York 1991*. Springer-Verlag, Hard copy. 77–96.
[100] Jim Woodcock and Jim Davies. 1996. *Using Z: Specification, Refinement and Proof*. Prentice Hall International.
[101] J. Woodcock, E. Gökce Aydal, and R. Chapman. 2010. The Tokeneer experiments. In *Proceedings of the Reflections on the Work on*, Cliff B. Jones, A. W. Roscoe, and Kenneth R. Wood (Eds.). Springer-Verlag, 405–430.
[102] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald. 2009. Formal methods: Practice and experience. *Comput. Surveys* 41, 4 (10 2009), 1–36.
[103] P. M. Woodward and S. G. Bond. 1983. *Guide to Algol 68 for users of RS Systems*. Edward Arnold.
[104] J. B. Wordsworth. 1992. *Software Development with Z*. Addison-Wesley.

Received January 2021; accepted February 2022