Medicine-by-wire: Practical considerations on formal techniques for dependable medical systems

Leo Freitas a, *, William E. Scott III b, Patrick Degenaar c

a School of Computing, Newcastle University, Newcastle upon Tyne, NE1 7RU, United Kingdom of Great Britain and Northern Ireland
b Transplant Regenerative Medicine Facility, Institute of Cellular Medicine, Newcastle University, NHSBT Newcastle Blood Donor Centre, Holland Dr, Newcastle NE2 4NQ, United Kingdom of Great Britain and Northern Ireland
c Electrical and Electronic Engineering, School of Engineering, Newcastle University, United Kingdom of Great Britain and Northern Ireland

A R T I C L E    I N F O

Article history:
Received 20 August 2019
Received in revised form 9 September 2020
Accepted 11 September 2020
Available online 28 September 2020

Keywords:
Medical devices
Automation
Formal methods
Certification

A B S T R A C T

We see the future of medicine as highly automated. Improvement in care-provision will be achieved by both increased clinician efficiency, as well as new computing assisted treatments and diagnoses. In other safety-critical industries, such as avionics and automotive, certification is dependability-driven. In contrast, medical certification is clinical-trial driven, which we argue will become increasingly problematic with increasing medical device and software complexity. By dependability, we mean the dictionary notion: reliable and trustworthy. Thus, failures are either avoided by design, or are accountable to a measured extent. This touches upon the verification (intent) versus validation (outcome) problem. Even though correctness does not imply safety, we do believe from our experiences that, the process of striving for correctness (verification) done right does shed light on safety; on whether the requirements/assumptions were addressed as intended (validation).

Medical device trials can lead to adequate assurances of safety, as defined by the local regulatory burden. Nevertheless, the nature of such complex systems means that certain errors may not be detected by trials and so additional efforts to reduce errors is needed. Our intent is, at least for software, to explore the contrast between approaches: correctness-by-construction versus correct-by-trial. Additionally, these levels of safety and effectiveness of systems vary across regulatory domains in different countries. A key challenge is how to achieve a successful interaction between verification tasks using formal methods and system development tasks within engineering teams without prior knowledge of formal techniques.

This paper describes a pragmatic process for the application of formal techniques, which is illustrated for three medical devices during pre-clinical development prior to certification. That means, the techniques are not only applied to realistic systems, but are also taken up by development teams themselves (i.e. cannot be entirely formal expert driven). We demonstrate differences in applying formalisms at the start, midpoint and final development stages. In particular, we describe the underlying socio-technical challenges and how we developed mitigation methods for each exemplar case. This paper is not about a general technique for medical automation, as we do not believe this is practical/possible given the varied/dynamic nature of medical problems.

© 2020 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

* Corresponding author.
E-mail address: leo.freitas@newcastle.ac.uk (L. Freitas).
1. Introduction

As highlighted by the recent Health Education England commissioned Topol report [1], as much as 75% of expert clinician-time is taken up addressing mundane repetitive tasks and analyses. This cognitive drain on a limited resource is creating healthcare delivery bottlenecks at the same time as the clinical burden increases with our ageing society. Furthermore, there are many clinical outcomes than can be improved through constant monitoring and adjustment of treatment. Achieving both of these scenarios would improve healthcare efficiency. Successful implementation of medical devices will require clinical outcomes with highly reliable and effective software. In this paper, we describe efforts to create, adapt and deploy formal methods to address these problems across several medical scenarios. This will not entail complete automation of medical device development, yet we hope it will help increase dependable medical device development in future.

Thus, in similarity to other safety critical industries, the healthcare industry is in demand of highly automated solutions to deliver tailored treatments. Already, control mechanisms in planes, trains and cars are not simply “analogue”. Built in electronic control, such as fly-by-wire, has been available for many decades.

In the healthcare domain, there is a similar “medicine-by-wire” direction. In this case, the objective is to simplify the intervention required by clinicians (improved outcomes), while allowing for much more finely tuned intervention (improved outcomes). It therefore has to adhere to precisely specified clinical-management and decision-making protocols, as well as triggering alarm states when healthcare status deteriorates in a dependable fashion. Increasingly, sophisticated control methods are being performed by software within medical devices, rather than via user manual control or intervention.

An important issue for all clinical systems is error reduction. With relevance to this paper, there are two considerations: user interface errors and run-time errors in the system software. Such safety breaches can cause both considerable patient and commercial harm [2]. According to a prominent advocate of better medical software safety: "If errors due to medical devices used in hospitals were considered a disease, they would be the third biggest killer after heart disease and cancer".¹

In the first case (for user interfaces), a large proportion of errors are related to user-mistakes for a number of reasons: training inexperience, fatigue, user interface confusion, as well as basic human error [3]. A 2008 study estimated the systemic cost of such errors to be in excess of $17.1 billion dollars in the US alone [4]. Considerable effort is therefore taking place to reduce the rate of user-based errors in clinical practice [3,5–11]. Still, there is a significant need for software to simplify the clinician’s user interface and attempt, where possible, to reduce errors (i.e. reliability needs to shift from the user to the system).

In the second case (for run-time errors), as systems become more complex, errors embedded by the development team during the design phase sometimes only become apparent during patient use. Healthcare systems are demanding ever more complicated software moving from simple automated systems, such as a syringe-pump delivering continuous doses of pharmaceutical therapeutics; towards complex devices delivering more elaborate sets of automated activities (e.g. lung function monitoring device collaborating with a respirator and heart bypass machines).

Such systems and devices are increasingly being relied on to perform complex tasks, including monitoring and actively intervening on therapeutic outcomes and closing the loop on the intervention. For example, closed-loop stimulus on advanced neuromodulation systems, such as brain pacemakers and neuroprosthetics, would like to modify their response based on patient circumstances. Existing high-end systems already perform cloud-based monitoring of patient status, adjusting treatment according to detailed medical protocols. This allows companies and clinicians to optimise stimulus programming. Future systems will allow for fully closed-loop control. To achieve these goals, a change to how algorithms and systems are developed and certified must be considered in order to avoid errors from occurring and to assure both patient and clinician confidence beyond trials, through a correctness-by-construction approach.

More surprisingly, a considerable amount of both morbidity and mortality may be preventable, if errors are rooted out during the development process. Nevertheless, the medical application space is perhaps a little distinct from other safety-critical applications. Medical devices try to treat one or more medical conditions and prevent both morbidity and mortality. However, if the patient deteriorates or even dies despite the best efforts of the device, that is an acceptable outcome. On the other hand, for other safety-critical applications this is clearly not the case (e.g. dying is not an acceptable outcome for flying). Medical devices have been applied to relatively simple tasks that are monitored on a regular basis. This is done by teams of doctors/nurses to ensure that their function is adequate.

Medical protocols and clinical guidelines are analogous to concurrent hybrid computer programs, with the patient’s state evolving as a function of time; and occasional shifts occurring due to unknown and known variables. For instance, considerations over (continuous) real-time and stochastic reasoning, in order to integrate, interpret and communicate adequate summaries to guide decisions on patient’s health. As medical automation develops to address more technically challenging tasks and manages mundane decision-making and patient care within these complex tasks, ensuring algorithms must adhere to medical protocols and clinical guidelines, becomes paramount to avoid errors that may cause harm. Typically, these errors only manifest under specific and unusual circumstances and as such may go undetected during clinical trials. In our approach, we perform both formal analysis of designs (i.e. formal specification and verification of intent), as well as specification-based testing of designs and code (i.e. provides degree of testing coverage).

¹ http://www.harold.thimbleby.net/mystery.html.
While medical systems are clearly safety-critical, methods from other safety-critical domains are not routinely applied to medical applications and processes. We will need to adapt and develop theories to handle these complex demands, while ensuring patient safety. Furthermore, we must ensure that when health-status shifts occur, that clinicians are alerted in an appropriate manner, while avoiding “alarm fatigue”. As such, clinical teams can continuously monitor and address the patient’s wellbeing. Our ambition is to raise awareness to these theoretical challenges and to establish a framework to solve current and future problems in healthcare automation: what we are calling “medicine-by-wire”, as the substitution or improvement of medical processes and treatments by computer systems and software.

In this paper, we present experiences in choosing and adapting formal techniques, as well as creating new tools, tailored for industrial medical application, development, and certification. A key motivation was to focus on which techniques could be effectively used by non formal-methods experts on realistic medical-problem scenarios. We highlight how to address socio-technical issues involved in three case studies, where involvement with formalisms took place at different stages of development/certification.

We focused beyond simply finding errors and creating specifications. As part of a cross-disciplinary socio-technical experiment, we also wanted to identify common ground between suitable techniques, which are accessible to stakeholders within their development environments with the least amount of disruption, and a “usual” development team. Decisions are tailored at aiding engineers who have little/no prior knowledge/training in formal techniques, and in positively influencing the gnarling medical device certification process by rooting out errors by design (i.e. applying correct-by-construction techniques in the development process) as much as possible.

Unsurprisingly, formal methods should be applied as early as possible in development to avoid wasteful replication of regulatory testing upon identification of errors. Based on our experience, there is substantial opportunity to improve the safety, dependability, and cost-effectiveness of life-critical medical devices through the adaptation and practical use of sound formal techniques to contribute directly to risk analyses and regulatory approval.

We aim to advocate the application of formal methods in medicine, influence future legislation, produce better quality medical devices, and create a methodology capable of achieving market standards. Our goal is to highlight the overlapping theoretical opportunities with real-world needs within the healthcare industry, enabling automation of mission-critical tasks and providing a framework to address a number of open problems likely to impact on the global healthcare needs for the coming years as advocated by Topol [1]. For instance, “to free specialist-personnel from mundane clinical activities and providing the gift of time” (i.e. new technologies increasing clinicians time for care and improved patient safety) to address the challenge of growing UK National Health Service (NHS) healthcare demands. That is, medical devices addressing such needs ought to dependably increase automation in medicine in our view. Even though this is a UK government report, we believe what it highlights is true of wider health care system demands.

In the next Section, we describe the background and related work in the area, particularly relating to the need for automated solutions to healthcare problems. Next, Section 3 frames and summarises three case studies we have worked/are working on, all of which are commercial and at different stages of development. Some details about each case is presented later in Appendix A. Section 4 discusses practical considerations of applying formal methods to real medical applications. Finally, Section 5 presents some insights, conclusions, and discussion on what we consider “medicine-by-wire” to be.

2. Background

In the decade following the 2008 financial crisis, the market for medical devices experienced significant growth driven by both social and technological developments. Now with the global COVID19 pandemic, the market is expected to expand further. According to a recent financial report [12]: “The global medical device market is expected to reach an estimated 410 billion dollars by 2023, and it is forecasted to grow at a compound annual growth rate of 4.5% from 2018 to 2023”. The major drivers for the growth of this market are healthcare expenditure, technological development, ageing population, and chronic diseases.

The UK represents 11% of the European market, has an excellent track record in introducing novel medical devices. It also hosts more small medical device companies than any country in Europe. At the time of writing, UK companies fall under the European Medical Device Directive and are about to switch over to the EU Medical Device Regulations (whose introduction has been slightly delayed due to the COVID19 pandemic). However, UK regulations may diverge in the coming years, as the UK leaves the European Union.

Medical automation

Medicine has traditionally relied on person-based interpretation and delivery of care. This has been managed via a hierarchical structure, where experts spend many years of training to understand nuanced data interpretations involved in care delivery. While high-level expert knowledge is required for complex and aberrant cases, large amounts of time is wasted managing routine cases. Predictable and repetitive cases could perhaps be better managed through automation of both diagnostic and therapeutic service delivery. Thus, a common vernacular is necessary to ensure that both the computer scientist and medical stakeholders can effectively co-create automated care-delivery systems.

As highlighted by Topol [1], up to 75% of clinical personnel time is consumed by “run-of-the-mill” diagnostic and therapeutic scenarios. This is due to the complexity of the data sets being considered, which require skilled interpretation.
Presently, monitoring is often performed by trainee doctors and reported back to senior consultants. This can lead to missing subtle data shifts, impacting patient care in cases where time is of the essence. Mistakes can also occur via errors in judgement, inexperience, bias or fatigue. Mitigation is currently performed via training and senior-clinician review. Yet, if such cases can be automated, it would unlock the potential for senior-clinician time to focus on complex cases. With improved efficiency, the quality of care will also improve within a given medical resource limit.

Technical challenges include: i) how to identify and characterise aberrant states precisely; ii) how to manage the flow of information presented in alarm-states; iii) how to make formal techniques relevant to certification; iv) how to prevent errors from user interface design issues; etc. These will ensure that adequate information percolates to care-providers without providing so much information to overload clinician’s time. Even worse, to generate a state of alarm-fatigue meaning that aberrant states requiring attention are missed completely.2

Regulatory environment

Regulatory standards fall into two essential requirements: validation of efficacy and safety, and development under quality management systems. The specific requirements vary between countries: e.g. the European MDR (Medical Device Regulations) and the FDA (Food and Drug Administration) regulations in the US. In the latter case, the FDA provides General Principles of Software Validation [13], with a detailed description of the processes available in [5]. Nevertheless, there are common international standards set by the IEC (International Electrotechnical Commission) and ISO (International Organisation for Standardisation). Specifically, IEC 60601, IEC 62366 and ISO 14971 deal with the functionality and safety from a risk management perspective. ISO 13485 and IEC 62304 relate to how the software is created and maintained within a quality management system. These are often non-prescriptive and open to interpretation. This approach is deliberate, given the full range of medical devices these standards regulate. Thus it is up to the submission team to present safety in terms of their risk management and quality management processes. Despite this, current EU regulation for medical software is open-ended, in comparison to FDA regulatory environment in the US. Nevertheless, there are processes in place to port certification (via IEC62304) for FDA approval, which further complicates scenarios.

In our experience of the process in the UK, risk assessment is typically done based initially on team experience and subsequently documenting further risks into the assessment. Development of the hazard analysis often follows “in house” rules (i.e. there is no prescriptively defined set of rules), many of which are derived from hardware design principles. Teams typically include physicists and/or electronic engineers, but seldom any computer scientist. As systems become more complex in nature and perform greater degrees of “independent” provision of care, such process may not be sufficient for software risk mitigation. As such, there is a need for formal documentation and analyses, all the way to code-level verification, for life-critical software.

Medical devices are comprising of increasingly complex embedded systems. Yet, in our view, software regulation in the UK [14] is not as strict as other safety-critical areas like avionics [15]. Part of this is due to the presence of many small-to-medium businesses with limited budgets generating innovation and an ethical imperative to ensure that healthcare innovations reach the market for the “greater good” as soon as possible. Nevertheless, as embedded systems become more complex and play a more prominent role in patient care, the likelihood of unanticipated and hidden errors to manifest will increase in probability. Similarly, as systems become increasingly closed-loop, the clinical consequences also increase [3, 7]. For instance, the FDA issues numerous recalls for manufactured devices, due to severe errors being identified posthoc [2, 16].

In the medical device domain, problems related to systems and software safety are often challenging to predict [3]. The multidisciplinary nature of the area, requiring knowledge of electronics, physics, clinical medicine, embedded software design etc. makes this challenging. The social and health benefits of safer health care are obvious. Nevertheless, adding yet another set of requirements from formal approaches can, if inconvenient, act as a significant barrier to teams trying to get devices through trials. As such, it is important that the implementation has minimal overhead on the R&D process of the medical system.

Cost-effective formal methods for medicine

The application of formal methods can reduce the possibility that software errors manifest in such medical systems; de-risking these complex systems. Specifically, formal modelling can decrease (or eliminate) certain classes of (run-time) errors, as well as help mitigate/control how errors could entail risk. Of course, this is the best-case scenario: it is possible to introduce errors by the misuse of formalisms too. Setting stakeholders expectations right about what formalisms can and cannot do is crucial to building trust and increasing uptake. For instance, even if a system’s software is automatically generated through full refinement chains, errors might still exist given that the formal interpretations of requirements might be incorrect/incomplete. To address this, accessible tools will need to be developed and brought to market to ensure patient-safety and ensure public confidence are maintained.

We have identified common ground between applications and the need for curating accumulated expertise across problem classes to avoid time-consuming and costly rework. To ensure active engagement with the medical community, we must

---

2 www.nytimes.com/2019/07/09/science/alarm-fatigue-hospitals.html.
consider which fields of medicine will most likely be early adopters of formal techniques. From canvassing experts within multiple disciplines (i.e. medicine, computing, physics, and electronic and biomedical engineering), we have identified areas with substantial automation requirements (i.e. the need to introduce devices to support clinicians or reduce their workload). These are associated with high-impact in medical treatment.

Focus on documented cost-effective safety and assurance is already part of other formal development processes: a US National Security Agency study has shown that formal automation techniques are both reliable and cost-effective at scale [17]. Other significant examples of successful industrial-scale application of formalisms exist [18,19,17]. Given the high-priority assigned to documenting quality and presence of limitations requiring automation, embedded medical devices are promising candidates for the development and application of formal methods to govern medical processes automation. Furthermore, to ensure engagement and integration, it is important to address a variety of socio-technical barriers including: cross-disciplinary vocabularies, patient view-points, and regulatory considerations across countries (e.g. UK MHRA, US FDA regulatory agencies).

Areas of medicine, most-suitable as potential early-adopters, include those with specific and burdensome regulatory requirements for quality and consistency. For example: life-critical (UK MHRA class III) medical devices. Other areas beyond devices, such as cell-gene therapy production, rely on “Good Manufacturing Practice” (GMP) [20] and require close adherence to detailed “Standard Operating Procedures” and protocols in version-controlled quality systems. To ensure adherence to such protocols, specific manual batch records are meticulously kept, which lay out and record all manufacturing steps performed by trained and validated operators; with key steps observed alongside for further validation by other expert staff. In certain safety-critical procedures, further validation by specialist qualified persons’ is required before a manufactured product can be released for clinical use.

Fields with such requirements include pharmaceutical production and some (GMP) transplantation procedures, such as Pancreas Islet Isolation. These regulatory requirements, while necessary to ensure safety during healthcare delivery, have dramatic costs and logistical impacts. For instance, problems from a single batch can impact on thousands of patients. Some cases, particularly commercial scale-ups, present challenges ripe for automation with both physical and resource-dependent limitations on personnel impacting real-world applications. This is a potential area to focus on in future. Here, we discuss the formal methods applied to medical devices only.

Related work

An excellent description of how regulatory bodies are influenced by evidence-based methods, as well as a survey of appropriate (informal verification and validation techniques, is in [5]. Application of formal reasoning to medical devices does exist; an excellent recent literature review covering both quantitative and qualitative use of formal methods to medical software is given in [21]. Some are quite successful, technically sound and relevant to industrial problems. For instance, Phillips Medical and Verum created a combination of BSDM and CSP for a number of complex devices [6,22,23]. This is highly commendable, yet necessitates adoption at the early stages of development, because various assumptions are needed on the structure of finite state machine (FSM) control and layout of various components. These are sensible assumptions, yet require specific development-team configuration (i.e. a trained software engineer/formalism expert) and demands significant upfront financial investment (i.e. tools are not free to use in commercial or R&D, which for SME is a serious limitation).

At the other end of the spectrum, high-quality open-source tool chains exist. These enable the application of formalisms from the capturing of requirements and risks, all the way to the source code with data refinement and proof support. An example is the application to a rather sanitised and simplistic dialysis machine, with abstractions to how such machines work in practice [24,25]. In fairness, these were papers associated with a case-study challenge-description prepared for the ABZ conference (e.g. it was not an exercise for a real dialysrer). The Event-B tool chain demands considerable investment to learn a number of alien languages to the non-expert. Moreover, these brilliant combinations of tools face an uphill struggle over regulatory processes, which presently may/do not recognise efforts. An interesting example of the application of another tool chain for ASM to dialysis is in [25].

An ambitious and successful attempt at applying formalisms that inspired our earlier efforts was the formal analysis of the Boston Scientific cardiac pacemaker [26]. To our knowledge, this was the first attempt to tackle the combination of applying formal techniques realistically for industrial-scale certified medical applications. A crucial difference to our efforts, however, is that the pacemaker work was a post-hoc exercise: modelling and verification were performed after certification had taken place, rather than during actual development. A tricky (yet shared in our experience) socio-technical issue happens if an error is found in such post-hoc analysis, given medical device recalls are complicated, and serious perception/financial/legal damage could follow. This example served as inspiration for our approach described in the brain pacemaker case study (see Appendix A.2).

The logic behind our style of presentation for the case studies in Section 3 and Appendix A is inspired by recent successful large-scale applications of formalism in industry. In [27,28], experiences of how and why certain formal approaches failed/worked is presented for the application of formalism in Facebook and Amazon, respectively. These examples considered formal technique choices and their adaptation according to actual practice within their corresponding development environments and teams in mind, while keeping stakeholders engaged. The socio-technical discussion for each case study was inspired by the US NSA agency study hints and warnings of how formalism can influence stakeholder successfully (or not) [17].
3. Case studies summary

The aim of our work to date with medical devices is to research and identify formal methodologies that could: be used directly, be integrated/adjusted in a realistic medical device development team or needed to be created from scratch. The process was as follows:

1. Identify industry-specific useful high-integrity development methods;
2. Experiment with chosen solutions within realistic scenarios;
3. Give stakeholders convincing evidence on adequacy and applicability;
4. Embed our results within the regulatory processes.

Our risk analyses were designed to satisfy regulatory requirements by presenting evidence for the technical file compiled as required by regulatory authorities (e.g. UK MHRA). Guidelines assessed the hazards associated with a medical device [5,14]. These hazards included potential hardware and software failures as assessed by design-teams and their regulatory experience. The developed risk assessment uses a combination of: requirements engineering, aligned with risk assessments; model based design, with model checking and theorem proving analysis techniques; data refinement, from model-based design choices (in VDM) to concrete implementations (in C); and source-code analysis in low-level programs and device drivers, for freedom of run-time errors, as well as functional (total/partial) correctness of low-level programs that use pointers and shared memory.

Our approach is language and tool agnostic: choices are based on team experience, adequacy of technique to the problem, learning curve requirements, and other practical considerations. The bottom line is to choose the “right” method acceptable to stakeholders that is adequate to the problem “warts and all”.

The case studies details are given in Appendix A on page 12. Here, we focus on the key issues and lessons learned. The case studies are on:

1. Neonatal (0.8...8 kg) dialysis machine for the UK NHS;
2. Combined gene therapy and optoelectronic brain implant to treat epilepsy;
3. Preservation control system for organ transplantation.

Their exposition is divided into four subsections:

1. Context: a description of clinical need and work environment;
2. System: a technical description of the actual system;
3. Verification setup: how the verification is organised, what needs and limitations this entails, and rationale for choices;
4. Results: summary of key results, both technical and socio-technical (i.e. how each controller solves the clinical problem, how the formalisation catches and prevents errors, etc.; development challenges, how teams perceive the process and how much expertise/time is required).

Our aim is to aid the reality of stakeholders, whom have no/little prior knowledge/training in formalisms, and to be of help (rather than hinder) the gruelling medical certification process. Each case illustrates the application of formal techniques at the end (Appendix A.1), the middle (Appendix A.2), and the beginning of the design process (Appendix A.3), respectively; and how that can affect potential outcomes or design decisions. We see this as important because it is rarely possible to start participation at the beginning of development given the nature of the industry (e.g. small businesses, commercial sensitivity, lack of expertise in formalism, formalism as a burden within regulatory process, etc.), and to demonstrate that the ideas we chose apply to any stage of development.

The work involved the identification, adaptation, and application of stable and cost-effective industry-standard formal techniques with acceptable learning requirements. We also create engineering solutions with formalisms in mind through auxiliary tools that are paramount to uptake of these techniques by practitioners given the size of problems. Some results exist as Newcastle MSc and undergraduate projects [29–35], yet their publication is delicate/restricted given NDAs, pending patent applications and the commercial nature of designs. Other results have been published [7].

The work focused on the formal modelling and analysis of key design documents, controller-components, and their software implementations. These controllers drive the medical activities (e.g. dialysis cycle, brain signals, organ preservation). They also deal with error and alarm management, both critical to deliver treatment and to ensure adequate user experience in intensive care units or surgery theatres (e.g. low alarm fatigue, fast and accurate/accountable summaries of current clinical conditions).

Table 1 (on page 7) presents a detailed summary of what we saw as key metrics associated with developments of each case study (i.e. NIDUS infant dialyser, CANDO brain pacemaker and POLAR organ preservation). The FSM (finite state machine) structure column gives the number of explicit states and events, which corresponds to explicit functionality versus hardware/software interrupts associated with each functionality. The FSM properties column tells the number of properties verified either through model checking or theorem proving (see also Isabelle/HOL columns on Table 3 on page 8). The C/C++ columns list the thousands of lines of code (KLOC) associated with each project. These C/C++ columns were: LIB-
KLOC refers to library dependencies and automatically generated hardware-related code; KLOC refers to the code base we verified; AKLOC refers to the formally verified annotated codebase (i.e. modified code with formal annotations); VCs refers to the number of verification conditions discharged by the C code verifier [36] and associated with the corresponding AKLOC. The VDM columns refer to the VDM models representing the abstractions from the C/C++ code in KLOC. The corresponding VDM proof obligations (POs) refer to the POs discharged using the Isabelle/HOL in the adjacent columns. The Isabelle/HOL columns refer to either ad-hoc proofs from specifically chosen eC VCs (Dialyser) or VDM POs of interest (POLAR) and translated VDM POs (CANDO). Finally, the interactions columns detail the number of versions in the development and the number of associated student projects.

The versions column in Table 1 (on page 7) refers to an interesting socio-technical aspect of the work. Each student project follow the style advocated in [37]: each formal design decision throughout the development that leads to a major version are carefully documented and explained. These versions, and their explanation/rationale, provides unique insight into the direction verification is taking, what lessons could be/were learned in the process, and why decisions were taken. These correlate to some of the lessons in Table 4. Each version details are described in student projects [29–35]. There are ongoing MSc and undergraduate projects to publish in the near future once IP restrictions are lifted.

Next, Table 2 (on page 8) presents an overview of the overall effort involved in each project, as well as how the verification teams were set up. For each of the case studies, the person-years and real-years numbers are based on our observations confirmed by each team. This at best gives an impression (rather than a predictor) to what efforts were. The verification person-years show that verification efforts were quite reasonable and limited in cost/time. The team configurations/split column shows who was involved. In all cases, it was mostly students and one verification expert over the period.

This information on team setup is at times incomplete (or even misrepresented). For instance, in the CANDO project, multiple institutions and disciplines are involved; some that have nothing to do with software, yet all of which are influenced by software problems. Having said that, from a human or even commercial perspective, the scale of the effort itself is far less important than the impact of the consequences. For example, if a baby dies from a preventable error manifesting in a dialyser, all certification effort is irrelevant. If software failure(s) lead to such consequences, it can sink the overall development programme including the efforts of the extended (formalisation) team. Such incidents erode confidence and can effectively kill a development programme, which can last for 10–15 years in the case of medical devices. Thus, the case to apply formalisms is greatly strengthen, both technically and commercially, in our view.

An interesting observation is that medical device developers/engineers largely resisted the use formalisms, whereas clinicians/patients welcomed it. We think this difference comes from the realities associated with medical device regulations. Developers are under tight schedules given clinical-trial conditions (i.e. device lock-down mid-way through development) and any effort that at first seems unrelated to regulatory approval, such as formal developments, are perceived as an unnecessary burden. On the other hand, clinicians/patients want the best device as an result come what may. This tension highlights a key challenge associated with medical devices regulatory processes.

Table 3 (on page 8) presents a summary of language and tools used. This includes techniques involved and what was their application to each case study. The list is not exhaustive, given numerous small-scale tools, models and experiments were attempted. For instance, some of the dialyser’s FSM properties were checked using PVS instead of Isabelle for technical reasons relating to MAL and its encoding of LTL. Moreover, the use on Spin in CANDO was rather limited and could be extended.

Finally, in Table 4 (on page 8), we summarise the lessons learned in general and in specific to each case study detailed in Appendix A. The general lessons apply to all three case studies and we think they are highly likely to apply to other cases too. Some lessons apply to multiple cases, depending on when formalisms were adopted. For each row, we qualified each lesson according to whether it related to model based design (MBD), medical device certification (Cert.), or tool support (Tools) issues and solutions. In POLAR table, the * indicates aspects we cannot yet discuss due to intellectual property protection issues.

4. Considerations in formalising medical devices

Next, we present a roadmap to what we consider to be a viable set of general principles and considerations for applying formal methods to medical devices and processes. Having said that, we do not believe the specific choices and details will be generally applicable, given the dynamic and highly complex nature of medical problems.
Table 2
Work x real time effort and team splits in each case study.

| Project | Clinical use      | Development team | Verification team | Verification expertise split |
|---------|-------------------|------------------|-------------------|-----------------------------|
|         | Person/years      | Real time        | Person/years      | Real time                   |
| NIDUS   | Neonatal dialysis | 4                | 0.25              | 2 years                     | 5                |
|         | 4 years           |                  |                   |                             | 1 senior + 1 expert |
| CANDO   | Brain pacemaker   | 25               | 0.25              | 2 years                     | 3                |
|         | 7 years           |                  |                   |                             | 1 expert         |
| POLAR   | Organ preservation| 2                | 0.5               | 2 years                     | 5                |
|         | 2 years           |                  |                   |                             | 1 expert         |

Table 3
Formal language and tool applied with related technique and context of use per case study.

1. NIDUS, neonatal dialyser (Late Adoption)

| Language | Tool | Technique                  | Application                                                                 |
|----------|------|----------------------------|-----------------------------------------------------------------------------|
| BDSM     | Verum| Requirements analysis/elicitiation | Formal specification of requirements to dialysis controller                |
| CSP      | FDR  | Refinement model checking   |                                |                               |
| MAP      | NuSMV| LTL symbolic model checking |                                |                               |
| VDM      | VMJ  | Symbolic simulation, code coverage |                                |                               |
| C++      | eCv  | MISRA compliance, resolution proof |                                |                               |
| HOL      | Isabelle | LCF\(^{\text{a}}\) theorem proving | Specific VDM proofs and eCv VCs                              |
| Java     | ANTLR| Parsing, translation, code generation |                                |                               |

2. CANDO brain pacemaker (Midpoint Adoption)

| Language | Tool | Technique                  | Application                                                                 |
|----------|------|----------------------------|-----------------------------------------------------------------------------|
| VDM      | VMJ  | Symbolic simulation, code coverage | Eliciting formal specification of CMOS instruction set [29]                |
| HOL      | Isabelle | LCF theorem proving | CMOS VDM model proof obligations [29,30]                                    |
| Promela  | Spin | Model checking             | CMOS main loop proof of termination [30]                                    |
| C        | eCv  | MISRA compliance, resolution proof | CMOS device driver static analysis and functional correctness [29]          |
| Java     | Overture | Parsing, translation, code generation | From VDM to Isabelle, Spin and C [30,35]                                  |

3. POLAR organ preservation (Early Adoption)

| Language | Tool | Technique                  | Application                                                                 |
|----------|------|----------------------------|-----------------------------------------------------------------------------|
| VDM      | VMJ  | Symbolic simulation, code coverage | Eliciting formal specification of organ preservation controller [31]       |
| HOL      | Isabelle | LCF theorem proving | Specific VDM proof obligations                                              |
| C        | eCv  | MISRA compliance, resolution proof | Organ preservation controller device driver static analysis and functional correctness |
| Java     | Overture | Parsing, translation, code generation | From VDM to Isabelle and C [35]                                           |
| Python   | Python | PID-control simulation and calibration |                                |                               |

\(^{\text{a}}\)Logic of computable functions.

Table 4
Important lessons learned in general and per case study, by category (i.e. model based design, certification or tools).

| Category | Lessons                                                                                     |
|----------|--------------------------------------------------------------------------------------------|
| General  | 1. MBD Changes percolated through models                                                   |
|          | 2. Cert. MISRA compliance simplified/reduced C/C++ VCs numbers and complexity              |
|          | 3. MBD Dependencies handling (e.g. libraries, auto. gen. code) was crucial/complex        |
|          | 4. Tools Bespoke (yet reusable across projects) tools required to enable development team engagement |
| NIDUS (Late Adoption) | 1. MBD Error detection and prevention linked to risk analysis profile (i.e. from Word file to Isabelle proofs via VDM) |
|          | 2. Cert. Verification directly linked with regulatory process activities                    |
|          | 3. Tools Late adoption entailed stricter/smaller set of options (e.g. start from the code) |
| CANDO (Midpoint Adoption) | 1. Tools Simulation and test coverage were important for hardware debugging |
|          | 2. MBD Verification setup was compatible with future project stages (i.e. 1 out of 3 control stages: optrode, optrode array, chest unit) |
|          | 3. Tools VHDL hardware-description library dependencies management was useful across scenarios |
| POLAR (Early Adoption) | 1. MBD Early adoption enabled clearer abstractions and simpler proofs associated with models |
|          | 2. Cert. Traceability was possible from requirements/patents to models, to design, through to the controller code |
|          | 3. Tools Simulation and test coverage was important for hardware debugging                   |
|          | 4. Cert. Training and exploration of unpredictable/unknown preservation scenarios           |
Practical considerations

As with other embedded safety-critical systems, access to the real-world rig conditions is costly (life/financially), has limited access, and is difficult (e.g. machine test during time-sensitive clinical-trials demanding limited extra hospital space/time). On the other hand, medical device certification requires a specific number of adequate usage within (real/ill) target patients. Testing is often difficult because the device is interacting with a complex physiological system under unknown and often unpredictable circumstances, where the cases the device use could fail may be rare.

This makes testing very difficult indeed. Nevertheless, it is clear that developing a risk analysis is essential when dealing with life-critical medical systems. Conventionally, risk analyses are submitted to the regulatory bodies prior to clinical-trials to generate evidence that requirements have been satisfied; and the device is safe for more widespread patient use. Trials then need to show efficacy at later stages. These are onerous tasks requiring substantial amounts of costly pre-clinical and clinical test data under special clinical-trial permissions. Emergence of failure late in development can be disastrous. Any significant redesigns/updates may require re-certification; delaying the process and adding cost. If failure emerges during a trial it may expose companies to financial and reputation liability, which may not be survivable for a company/product line depending on the size of the company and severity of failure.

Regulatory process

Medical device standards require a prescribed documentation of risks and associated mitigating measures to be taken against use of devices. That is, to keep such risks as low as reasonably practicable, or at the very least to demonstrate awareness and potential mitigation strategies are in place. Unfortunately, when dealing with software, however, risks may not be obvious even to the most experienced development team. We believe that formal verification of control software is a necessary requirement for (UK MHRA) class II or III medical devices (i.e. those with highest-level of risk to life).

Another key aim is to push regulators and thus regulations towards allowing evidence provided through formal reasoning to be accepted alongside clinical-trial data for software validation. Even if there has been a misunderstanding in the formal modelling choices with serious consequences in practice, formalisation enables the traceability of such situations when/if they happen with traceable consequences. Moreover, it would be greatly beneficial if safe language subset restrictions like MISRA were common practice, if not mandatory. This is inline with other safety-critical industries, such as the use of formal methods in DO-178C [15] for avionics or MISRA-C [38] for automotive software compliance. We are holding ongoing discussions with the several notified bodies and the UK regulatory body responsible for medical device compliance (MHRA).

We hope that by generating a deeper understanding of the benefits formalisms offer, that these methods can be adopted into the certification pathway as has happened in other safety-critical industries. Ultimately, the hypothesis is that this would motivate adoption by commercial medical device companies and improve medical safety and patients’ health outcomes.

We see the use of formal verification for the absence of certain erroneous states, as opposed to documenting safety (designed to meet regulatory demands) for any software automating a task, which may expose patients to imminent harm. The key point here is to highlight the existing standards inadequacy to promise this in a rapidly evolving world, where software is being relied upon to do increasingly more complicated tasks for both patients and clinicians.

Clinical realities

Our work is enabling us to understand how to mathematically document the realities of data processing and management in medicine, in particular where a general audit capability can be leveraged across multiple applications. For instance, a dialysing or an organ preservation machine are fundamentally similar control systems: they keep a particular curve of flow and pressure within specific windows; they provide summaries to where the “patient” is in the treatment process or the cumulative effect on any adverse event over periods of time. For instance, in dialysis overpressure could collapse a patient’s vein because of blood being drawn too fast, or in organ preservation excessive pressure could damage organs.

These known-unknown and unknown-unknown conditions present an interesting (and novel) challenge to the application of formal techniques. Where documented safety is of paramount importance, further measures to document the absence of potential classes of risks through application of formal techniques ought to be considered.

A difficult issue is timing. Late discovery of issues can lead to potential re-trials, which do create a socio-technical conundrum: if applied post-hoc, formalisms can lead to complex practicalities. If errors are found in embedded software close to certification, how can you avoid re-trial/starting over? Costs and certification burdens aside, is delaying the process the right thing to do (i.e. holding a new treatment from the market could be more damaging than rolling it out as patients not receiving treatment may die)? Our experience is that, eventually once in the real-world, any such potential risk will manifest in practice given enough opportunity, making this an even more difficult task. This is not new: other safety-critical systems suffer the same problems; yet the medical device domain is a very different reality.

In [39], the authors argue whether the Boeing 737 MAX was really a completely different plane made to look like a 737 for training and regulatory purposes to disastrous consequences. In the design of a new system for trains (e.g. London or Paris driverless trains) or cars (e.g. Bosch’s new start-stop or cruise control systems), delays can be costly and damage reputation, yet are preferred to recalls or actual fatal accidents. For medical devices, on the other hand, delays because of the application of somewhat “alien” (formal) techniques to engineers, delay development preventing saving lives earlier,
Despite medical devices being similarly safety-critical, given their highly unpredictable working environments affecting a large number of people.

Such catastrophic error scenarios are rare. Thus, the balance seems in favour of pushing development through as quickly as possible in order to save lives, despite the potential for software errors causing harm in such rare situations. In fairness, we think this oblivion to certain kinds of risks of software is already present in various medical certification processes worldwide, and that has to do with the culture of clinical trials as the gate keeper of safety and efficacy in medicine. Unfortunately, in our view, clinical trials alone are not good enough for preventing errors in complex software. This view is shared by the notified bodies we have had discussions with.

Another complex reality of this industry is that commercial development of many devices is done through small/medium enterprises, where the financial incentive for years of design and development is in recouping costs through disposable kits (e.g. the syringe and fluid lines in the dialyser). That means budgets are already strained and that further “unnecessary” activities, such as formal design or code-level verification, are simply prohibitive, particularly given they are not taken into account in the certification process.

We faced these difficult realities in all case studies discussed in Appendix A. This led us to think that, despite many medical devices clearly being safety-critical in nature, directly applying formal techniques would require considerable development both in engineering (e.g. easier tools and languages) and theoretical (e.g. reusable summaries accounting for real-time, concurrent, stochastic and probabilistic features) terms.

Development teams

Nevertheless, if formalisms are applied as early as possible, and there is plenty of evidence for it being beneficial and cost-effective in other industries [19,17,18], outcomes can be greatly improved. A key proviso though is the team “expertise-splitt” [17]: required expertise must entail realistic teams of experts (and junior, entry-level) engineers. In our case, we had the chance to experiment in practice with late, midpoint, and early development stage adoption; with teams including senior engineers, trainees, students and some formal methods experts (see Table 2 on page 8).

Development teams for medical devices often comprise a combination of clinicians, biomedical and electronic engineers and physicists. Given limited budgets and long time-frames with diminishing returns, investment in software engineers (never mind formal methods) is often neglected. This is particularly true in Europe, where medical standards for software do not focus on correctness or dependability. Having said that, examples of good software engineering practice for realistic devices, as well as the application of formal methods for medical devices exist [6,22,23].

As part of our socio-technical experiments, we varied team configurations to understand what would and would not work. Evidence so far shows that a formal methods expert has to be intensely involved at first in order to ascertain what needs to be done where. Researchers and students with software engineering and/or formal methods backgrounds are capable of handling guided verification tasks independently to great results, cost effectively. This is demonstrated by the case studies’ results show in Appendix A. This is inline with experiences of applying formal methods in other safety-critical areas [17–19].

Applying formal methods to medical problems

To succeed, the choice of formalism(s) to be applied is a crucial decision. Ideally, the same collection of formal tools and techniques could/should be used across examples. In practice, however, this is unrealistic given the varied nature of teams’ expertise, time-frames and cost-limitations involved.

Crucially, whatever methods are chosen, they ought to be taken up by the engineering teams themselves. Otherwise, all the effort will be an academic exercise. Therefore, our approach in each case was to choose the most suitable method that would cater for the underlying needs and that would be within the team’s expertise with limited training.

For instance, for the dialysers case study (see Appendix A.1), we used a combination of: BSDM as an attempt to formally capture requirements; CSP/FDR and MAL/LTL/nuSMV, to encode the system’s FSM design, where each identified hazard profile risk was encoded as a property to be model checked; VDM to discover and encode rich data-type invariants and functional properties (pre/post conditions), and to enable symbolic simulation and code generation to C++; and eCv for code-level functional correctness verification [7]. The use of BSDM and CSP failed [32], as it imposed a more component-based approach not suitable for where the dialysers’s design was, hence we changed to MAL/LTL [32]. This was important given the complex nature of the FSM and the properties being checked. The use of VDM was important to discover specification and help with the low-level C++ verification with eCv.

For the brain pacemaker (CANDO) case study (see Appendix A.2), the FSM architecture was much simpler than the dialysers and with simpler properties. This meant we could afford to take a theorem proving route and actually prove that properties of interest held under the now identified missing assumptions. We used a combination of VDM, Isabelle/HOL and eCv. VDM enabled discovering and documenting FSM invariants, performing symbolic simulations and code/specification coverage [29]. We translated this to Isabelle/HOL tools and proved properties of interest [30]. Finally, the verification of the C code ensured that the FSM control code was correctly implemented according to the VDM specifications, and that low-level bit-vector expressions were simpler and correct [29].

L. Freitas, W.E. Scott and P. Degenaar Science of Computer Programming 200 (2020) 102545
For the organ preservation machine case study (see Appendix A.3), we followed the same approach of identifying the right technique for the task within capabilities of teams involved. For IP-protection reasons, we cannot disclose more details at this point (see details in [31,35]), but we hope to do so in future publications.

Obviously, any formal models and analysis are an approximation of reality, as much as possible. Thus, they do not entail freedom from errors altogether. They do entail, however, a controlled and dependable development process, where design decisions can be mathematically documented and hazards/risks/properties of interest can be checked against these mathematical documents. We see these applications of formalism as a step further, building on the use of hazard/risk analyses alone: they mathematically encode design decisions as much as possible. Moreover, as the system evolves, formal modelling has also proved invaluable in accounting for the consequences of change. Therefore, even though the complete eradication of error is not usually possible, we see formalising medical systems as invaluable. We highlight various examples of such situations in our case studies in the Appendix below.

5. Conclusions

In this paper, we have detailed our experience to date on applying formal methods to medical devices and the tangible real-world impact that such application may have on medical device development. These efforts cover only a tiny fraction of the potential for this area, both theoretically and in other application areas involving automation of medical tasks (i.e. medicine-by-wire).

Our efforts have demonstrated the need for the application of formal methods in device development to avoid dangerous hidden errors from manifesting in a clinical setting after certification. We have shown (see Tables 2–4 and Appendix A) that this should be done ideally as early in the development process as possible to avoid wasted resources during testing or design modification impossibilities because of the regulatory process. Still, later or post-hoc application can produce positive impact.

We have also presented real-world considerations to the application of formal methods to medicine, including its socio-technical considerations, and demonstrated how existing theories may be modified and developed to suit complex medical devices.

In our experience, the overall answer is that a well-balanced team, with adequately chosen/adapted formalism can (and has) delivered real benefit. This is both in terms of: life-critical error discovery, prevention of future errors developing or being hidden in the designs; and link with the certification processes, by embedding formal results in the ongoing certification efforts for the case studies we have been working on [7,30,29,35]. Ultimately, we would hope to influence certification processes to take these results into account: this would improve dependability of medical devices, as well as alleviate some of the activities associated with certification.

Our aim with the work presented in this paper is to highlight this need to the formal methods community, and hopefully create enough momentum and enthusiasm to use, adapt and invent formal techniques to improve the quality and safety of healthcare, like formal reasoning has done to so many other safety-critical industries [18]. Some formal methods work for medical devices does exist [22,26,40,41], yet that is far from common practice, and is not taken into account in certification processes.

This is a realistic aim, with good supporting evidence. In [18], earlier applications of formal methods to different industries is discussed. More recently, examples like Tokeneer [17] and sel4 [19] demonstrate how formalism can be cost-effectively applied to the extent of beating any other approach in terms of error discovery and correction capabilities for both safety and security concerns, all within cost predictions in terms of time and money. We took inspiration from these projects to: spread your team according to “realistic” arrangements, time log activities, provide traceability capability between requirements, designs, implementation, etc. We focused on the correctness-by-construction argument with efficacy and safety (as well as security) in mind. This was alongside the (socio-technical) difficulties of applying formal methods during ongoing projects, whilst having to handle certification that did not recognised such (rather extraneous for the involved teams) efforts.

Future work Further work will be required to effectively extract nuanced data interpretation from clinicians. Development of new theories is needed to handle the real-time data interpretation required by the field, to safely automate clinical management. This theory of summaries is under development and is being applied to commercial medical applications.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We are particularly grateful for the reviewer’s key criticisms, which helped improve the paper considerably, both in technical accuracy and readership. For their real and significant patience and care, we are immensely grateful.
This work is partially sponsored by EPSRC STRATA platform grant EP/N023641/1, MRC Proximity to Discovery (MC_PC_17198) Grant, Wellcome 096975/Z/11/Z, and EPSRC: 102037/Z/13/Z.

Authors are grateful for the support from their corresponding institutes at Newcastle University School of Computing and Engineering, Translational and Clinical Research Institute and Newcastle NHS blood and transplant centre (NHSBT). We are also grateful for the considerable discussion and interaction with team members from: Newcastle Hospitals Regional Medical Physics; and CANDO and POLAR engineering teams. We are also grateful to the Blood and Transplant Research Unit (BTRU) in Organ Donation and Transplantation at the University of Cambridge in collaboration with Newcastle University and in partnership with NHSBT.

Various tools used were extended, thanks to the VDM development team at Aarhus University. We are also grateful for discussions on VDM with Cliff Jones, Nick Battle and Peter Tran-Jorgensen, and discussions on eCv with David Crocker. Various students helped with the case studies during their undergraduate and MScs projects at Newcastle. Finally, thanks to Nina for inspiration to work with medical devices dependability.

Appendix A. Case studies details

In this appendix, we detail the application of principles and practices discussed in the paper across three different medical device applications at three different stages of adoption during the development process.

A.1. NIDUS (Late development-stage adoption)

The “Newcastle Infant Dialysis and Ultrafiltration System” (NIDUS) is the first medical device capable of dialysing premature babies fast (i.e. 50 ml/hr) [42]. Dialysing small (0.8...8 kg) babies is challenging for many reasons: vascular access for haemodialysis in existing machines is problematic, as the size of the central venous line required for adequate blood flow is disproportionately large for the size of the baby. Moreover, existing dialysis machines are not accurate enough at managing critical parameters, such as low-volume ultrafiltration (water removal), and usual dialysis kits require too much blood volume to be viable for a premature baby. Thus, there was a clear need for a specialist device without risky off-label adaptations, such as blood priming. To our knowledge, prior to NIDUS, there was no dialysis machine approved for use in such infants.

In this case study, we started working at the end of development, close to certification, which limited what could be done. For instance, [32] showed how a modular approach to the handling of the three syringes in the original design (see Fig. A.1) could greatly simplify model checking of its finite state machine (FSM). Nevertheless, such redesign was not a viable option given the late-stage in development when we were brought into the project.

We modelled the control and information flow within the control software and between the software and various physical apparatus within the device (e.g. valves, syringes, filters, pumps, etc.), and the relation of software variables with the external environment (e.g. a baby, transfusion setup, medication, etc.). This includes the communication protocols between the software and the hardware devices, and between the already developed software and the user interface. The work involved formal modelling and analysis of the requirements into a design compliant with certification procedures [14], verification of implementation strategies and resulting products and deployment [33].

Our work identified serious potential flaws not detected through risk-analysis or clinical trials [7]. That is despite the system having run for a sufficient number of hours/patients, without incidents relating to safety or efficiency issues, hence they had evidence for certification. This included a very detailed hazard profile and multiple safety considerations. Despite that, the identified errors would have happened in practice, had the circumstances occurred.

Critical embedded software is often written in C or C++. From the point of view of verification and safety (i.e. code satisfies a given specification), C is a language not designed for verification, with many constructs that can give rise to undefined, unspecified or architecture-dependent behaviours. These could lead to the same code handled by multiple compilers for different target architectures potentially having different behaviours (i.e. concretely for the dialyser, negative numbers involved in bit vector logic and arithmetic). To mitigate these and other characteristics of C and C++, language subsets are widely used in developing high-integrity software, such as MISRA-C [38,43] or JSF [44].

For instance, several static-checking tools are available to enforce compliance to a greater or lesser extent. Well established high-integrity C/C++ subsets exist [38,43,44]. They are based on sets of rules that prohibit the use of various features and impose certain programming styles. We verified all the C/C++ code contracts defined for the dialyser controller using the Escher C/C++ verifier (eCv) [45], as well as achieved MISRA compliance, where technical details are described in [7]. MISRA compliance is used within various safety-critical industries, and it enforces a number of safety rules for C/C++ programs.

System

NIDUS consists of three syringe drivers, valves, a bubble detector, filters, and so forth as depicted in Fig. A.1 (top-right). The dialysis process has four stages: start, reset, wash and dialyse. The start stage differentiates which operational mode the machine will start in, which is either: “cold-start”, with a new dialysis kit; or “warm-start”, where the machine proceeds directly to the dialysis stage (bypassing the wash stage), if dialysis has to be restarted for the same patient on the same kit. The reset stage makes sure that all mechanical apparatus are in their expected positions and powered up. The wash stage runs saline solution through the device in order to prime/clean the dialysis kit. Finally, the (main) dialyse stage performs a
three-step control on blood flow that withdraws blood from the baby, filters it according to given parameters, and returns it back to the baby. Various components communicate via a serial bus, which in turn interact with the device’s controller representing the overall behaviour of the machine.

The dialyser is composed of hardware devices controlled by embedded firmware software written in C. A controller written in C++ drives the machine, and communicates with the graphical user interface written in C#, which displays current parameters, informs the controller of user key-presses, and animates the physical processes taking place with the involved hardware components. The control software also detects and warns of error conditions that require attention, as well as issues hardware interrupts that prevent the machine from behaving in a dangerous manner according to its risk profile (e.g. a power cut happens immediately if a bubble is detected, independent of software). The controller’s code verified is about 4.5KLOC of sequential C and C++ code, whereas the 7KLOC of C# for the user interface code was not considered.

**Verification setup**

The dialyser’s design was defined by a consultant nephrologist, physicists and biomedical engineers using a finite state machine (FSM) encoded as a CSV file. This described the instructions to the control software in order to perform the various message exchanges between hardware and software components.

Inspired by [23,46] and its successful application of formal methods for medical devices, we used BSDM/CSP within the (commercial) Verum tools to encode the state machine for analysis of various safety properties, as well as to explore how the underlying CSP could be extended and analysed with the FDR model checker directly [32]. These safety properties were derived from the risk assessment document that features in the certification process (i.e. a spreadsheet of risks and mitigation measures categorised by various criteria). Unfortunately, given the component-driven requirement of the CSP encoding of BSDM within Verum’s tools, it was counter-intuitive to model (as in the original FSM) three separate syringes’ interaction, rather than a single syringe used three times. This highlighted the “flat” nature of the original FSM, which was also state-rich, something that can limit the use of FDR. The resulting models were interesting [32], yet very different from the dialyser. This is why discoveries could not be taken into account, as we effectively ended up with a different FSM.

Next, given this flat design and its somewhat state-rich features (i.e. each state embedded 9 different bit-vector encoding of various hardware and software commands and parameters), we tried a different approach using modal action logic (MAL) [8] with properties from the risk profile encoded as linear temporal logic (LTL). We used symbolic (rather than explicit state) model checker NuSMV\(^3\) in order to tackle the state explosion and to enable temporal-logic model checking.

---

\(^3\) [http://nusmv.fbk.eu](http://nusmv.fbk.eu).
rather than FDR’s refinement checks. This was important because such characterisation of properties fitted better with the existent risk assessment within the regulatory process. Together with colleagues working within the CHI-MED project, which applied formal methods to numerous medical devices looking for human interface and safety issues, we built tools to streamline the encoding process. These tools enabled the encoding of the FSM in various target formats and its maintenance as it evolved given its size (e.g. 120 states, 32 events, and 7 and 5 hardware/software signals per state, see Table 1 on page 7) and properties to be checked (e.g., 25 safety properties from the regulatory documentation risk assessment related to the control software).

From the FSM CSV-file source, we built translators to NuSMV, VDM, and C++ [34], each of which embedded a particular strategy to capture the various invariants implicitly expected from the FSM and the C++ code, but that had never been explicitly stated anywhere. The NuSMV model of the FSM is used to model check whether risk assessment safety properties encoded in LTL were true of the FSM encoding; details can be found in [7]. The VDM model of the FSM [34] was used to symbolically simulate dialysis sessions and calculate test coverage (e.g. was the FSM completely traversable). The use of VDM was also crucial to help understand what type-invariants and pre/post conditions existed for the various FSM commands encoded in the C++ device driver, given that eliciting such invariants directly from C++ is error prone (e.g. C++ details and libraries are mixed within key abstractions that we were hunting for).

Another important aspect of the code verification setup was the identification and separation of features not directly representative of the dialysis or the FSM, but that were crucial “glue code” (e.g. OS-dependent libraries, C++ stdlib, IO communication, etc.). Inferring the minimal necessary contracts for these libraries, as well as shielding the dialysis-related code from such interactions interference, is an important part of the verification setup: these minimal contracts axiomatise assumptions about these dependencies. This approach can be repeated across projects, where such language and OS dependencies may overlap. Finally, the C++ code of the FSM provided boilerplate specifications (taken from the VDM experimentation) and automatically generated code (e.g. FSM representation through structs and classes, and bit vector manipulation libraries for various event handling and hardware signalling mechanisms) used in our code-level verification.

For the code analysis to be feasible, given its combination of C (for firmware device drivers), C++ (for FSM control loop), and C# (for user interface link with FSM), we first made the code MISRA compliant, without modifying its structure. This was important to help the chosen code verifier. Given the machine was already undergoing (CE-marking) certification and having had a significant number of hours of use with sick patients and without failure, structural changes without concrete evidence of errors were unrealistic.

We tried a number of tools: Microsoft’s, VCC5 Frama-C6 and Verifast.7 These tools focused on either concurrent C and/or memory safety with respect to pointer usage. Given we had a mix between C/C++, that the embedded code makes little use of dynamic memory, and that the dialyser was sequential, we judged that these tools were not suitable at the time (2014/15). Moreover, these tools did not check for MISRA compliance, a strengthening argument that we are pushing to become part of the certification process with regulatory authorities in the UK, given that it’s mostly automatic and requires no code contracts or formal methods expertise.

Although language subsetting for C and C++ makes for easier verification and avoids some common sources of error, it does not lead to correctness: we needed code contracts. These contracts are ultimately an interpretation of the informal requirements, hence we get to the verification x validation problem: we can build the system right, yet we might not build the right system. This can only be mitigated by involving users and through clinical testing.

We have used the Escher C++ verifier (eCv) [45]. It combines design-by-contract formal verification with MISRA compliance checking and encompassing memory safety and functional correctness of given specifications. It copes with C++ features including classes, templates and varied casting operators. Functional correctness can be verified using pre/post conditions, as well as type invariants given to classes, types, and functions. These contracts are written in an extended C++ language, including quantified expressions and mathematical data types, “ghost” (i.e. specification-only) code, and refinement relations. The tool ensures that the code for the chosen compiler and target architecture delivers the given contracts. Verification conditions (VCs) are proved using a combination of term-rewriting and a resolution/para-modulation automated theorem proving. eCv provides a detailed and structured-audit proof-trace, which is again useful for the purposes of certification. That is because eCv is already used for the certification process of safety-critical systems in other areas, such as military avionics systems. More importantly, from the point of view of non-expert user interaction, invaluable features include detailed contract suggestions upon verification condition check failures (i.e. counter-examples were already described in terms of suitable contracts). Some simple (yet common) contracts are also automatically suggested. These are important for non-expert users.

Results
For the resulting FSM design, model checking of the risk assessment properties revealed interesting issues, some of which were serious [7]. There are two starting modes: cold and warm, where the former imposes priming/washing the tubing

---

5 http://research.microsoft.com/en-us/projects/vcc.
6 http://frama-c.com.
7 http://www.cs.kuleuven.be/~bartj/verifast.
8 https://www.eschertech.com/products/ecv.php.
circuit, whereas the latter can start dialysis directly. It is possible to have a cold start (e.g. kit washing prior to) dialysis, followed by a (warm) restart (e.g. tubing or syringe readjustment for the same patient). If such complicated setup was the case (i.e., “cold” followed by “warm” start for the same patient), then there were conditions when unexpected states were reachable.

This could lead to debris being sucked-into the blood line after cold-start circuit-washing, but before dialysis, to be aspirated into the blood stream, if such warm re-start situation occurred and if the user did not perform a specific task. Ultimately, this unsafe situation ought to be ruled out by design. We did test the situation and the machine (mis-) behaved as expected when the scenario occurred. It is also relevant to say that this had never happened in practice before (and has since been fixed in the design), despite going through the necessary clinical safety trials with premature-baby patients prior to the identification of this error state. In our view, this highlights how software can cause unexpected outcomes that current regulatory practice might miss. Technically, we made the design satisfy the property that certain states could only be reached if combinations of states have happened in the past, hence ruling out this (mis-)behaviour by design.

More importantly, once the verification setup was in place, it was possible to easily iterate (and percolate) through various design documents. For example, an FSM change would trigger a NuSMV or VDM change, which we could model check or simulate. Then, when risk assessment properties changed, that entailed an extra/modified property check in NuSMV or VDM simulation runs. Finally, the C++ data structures representing the changed FSM state/events could be immediately regenerated and recompiled (e.g. they were type correct and MISRA-compliant by design).

The resulting controller code, automatically generated directly from the FSM design, is MISRA-compliant with respect to the rules implemented in eCv (v.6.10.6). Various coding issues were found; in particular those related to signed bitwise operators that, if compiled to different target architectures, could create serious problems with bit-vector interpretation of signed values [33]. These issues could potentially allow for dangerous behaviours, or inaccurate handling of error situations. Key findings included eight potential bugs in the C++ controller code, two of which could lead to serious outcomes for every dialysis-related function, for some target architectures. These bugs were not manifest in the target architecture chosen for the pre-commercial prototype, yet would manifest for other target architectures, as experiments have shown [33]. The contracts for these dialysis-related functions highlight the potential issue of signed integer overflow under certain bitwise and arithmetic operators, and improved the robustness of the code over time. We also revealed 50 MISRA-related issues in the code, such as stricter use of integer type-signage, as well as 20 minor issues like naming conventions.

The annotated and slightly modified code totalled 7.5KLOC, a not unusual increase from the original (4.5KLOC). eCv generated 782 VCs, of which 721 (92%) were discharged automatically and relatively quickly (e.g. under a couple of minutes). Of the remaining 61 (8%) unproved VCs, 38 (62% of unproved) the eCv theorem prover was initially not strong enough to discharge and an audit proof-trace failure was given. These were mostly related to bitwise expression patterns that eCv had never seem before. Another similar case was related to floating-point numbers. Some of these difficulties led to improvements in the eCv prover itself, which decreased this number of unprovable VCs from previous attempts at the problem. This does happen in practice, and helps improve tools, as they venture into new domains. The final 23 (38% of unproved) VCs were characterised as “too hard”, which means that given the right parameters for the prover heuristics and the various timeouts chosen by the eCv user, they might be dischargeable. Of the remaining 12 unproved VCs after heuristics tweaking (52% of “too hard” VCs), we categorised them into three families and used the Isabelle/HOL theorem prover to discharge them, whenever the designers thought they represented something crucial. The link to Isabelle was not too difficult because the residual VCs were C-independent and libraries for handling bit vector arithmetic exist for Isabelle.

**Socio-technical.** As a complementary (socio-technical) experiment, after isolating one key function that represented the main issue of signed bitwise operations in the codebase, we setup an undergraduate student project tasked with finding if there was any problems with the code by using code-level verification [33]. This was interesting as the student had neither seem nor used formal reasoning or code-level verification before. With only a few meetings (e.g. 30 to 45 minutes every other week) over the course of 3 to 4 months, findings were the same as in the original work done by the formal methods expert.

The development team, mostly experienced industry physicists, were friendly skeptics: once they understood the idea that we could mathematically represent the FSM and the assessed risks, they quickly realised what we could do to check whether the design and its changes were okay with respect to expected encoding of risk assessment issues from the regulatory documentation. This realisation was only possible when we could “empirically demonstrate” outcomes: no amount of explaining about the techniques was convincing enough, until they “saw” the results relating to what they cared about. An engineer anecdotally observed on the process and its demands: “the proof of the pudding is in the eating”. This reminded us of J Moore’s observation in one of his talks about formal verification: “we have to eat our own dog food”, if we are to have a chance of having industry take up our methods!

Many of these results featured in the final (UK MHRA CE-marked) certified dialyser (mid 2018), and some of the results technical details have been published [7]. A number of results and tools, however, are yet to be written due to time and IP protection limitations. Certified intellectual property (IP) has been bought, and the machine is now in production and in use within the UK.9

---

9 [https://allmedgroup.com](https://allmedgroup.com).
A.2. **CANDO (mid development-stage adoption)**

The “Controlling Abnormal Network Dynamics using Optogenetics” project\(^{10}\) is a clinically orientated project to develop a new form of brain implant. Its aim is to utilise a combination of gene-therapy and optoelectronics to provide closed-loop therapies to aberrant neurological conditions. The first target condition is being developed for focal epilepsy, which affects millions of people worldwide [47]. It is a multidisciplinary project involving electronic, chemical and material engineering, computer science, medicine and microbiology [48].

Their approach is radically different to current neuromodulation therapies, which act as either open-loop pacemakers or attempt to provide a single burst of electrical stimulus at the onset of a seizure. Instead, the objective is to continuously control the brain state of the location of brain tissue where the seizures begin (seizure focus), to prevent it from operating outside a safe domain. To address this, CANDO utilises a gene-therapy technique called optogenetics, which makes brain cells sensitive to light.

This derives from a discovery in 2003 of a protein called channelrhodopsin [49], which can be genetically inserted into cells to make them light sensitive. The great advantage of this technique is that it can be possible for different types of nerve cells to be sensitive to different wavelengths of light. Furthermore, as the stimulus and recording modalities are different, brain function and focal epilepsy prevention can be achieved without crosstalk, allowing for real-time closed-loop control. Crosstalk is the unwanted coupling between signal paths, such as electrical coupling between transmission media or capacitance imbalance between wire pairs, non-linear performance, voltage and/or capacitance coupling, and so on [50]. At the time of writing, there are two trials underway ([ClinicafTrials.gov](https://www.clinicaftrials.gov) Identifier: NCT02556736) for the restoration of vision in a subset of the blind, but they are yet unpublished.

All new techniques, of course, create challenges. In this case, optical pulses have to be generated in the brain. That requires local circuitry, which is to put all electronics inside a relatively large hermetic can. This local circuitry has to operate with minimal power to ensure that there is no heating of brain tissue, and that the stimulus needs to be optimised to ensure that photochemical damage is minimised. Exceeding safe-limits may lead to increased inflammation, reduced efficacy, and potentially permanent damage to the target brain tissue. It, therefore, is vitally important that commands to the brain unit are interpreted correctly and that its operation does not exceed safe operation.

In this case study, fast-paced milestones across multiple sites (e.g. Imperial College London and Newcastle University) involving multiple disciplines (e.g. physicists, electronic and biomedical engineers, material scientists, etc.) meant complex electronics being manufactured from scratch were difficult to formalise, given limited documentation. We started participation just before the middle of the project development (2016). There is still considerable work to be done for embedded control systems within the optrode array and chest unit (see Fig. A.2). Both rodent (rat, mouse) and non-human primate trials took place in 2019 – 20. Human trials are expected in 2021.

**System**

The part of CANDO for our attention consists of four optoelectronic arrays of optrodes that are implanted into the brain, and an external chest module connected via wires to the implant that controls the overall system behaviour [51–53] (see Fig. A.2). Each optrode contains multiple electrodes and light emitting diodes (LEDs), all of which are controlled by a specially designed complementary metal-oxide semiconductor (CMOS) chip with a 32-bit word bus and 34 explicit control commands (e.g. specific commands for switching LEDs on/off, electrode recording sites switching on/off, diagnostics, etc.). The array, in turn, controls each of its optrodes (e.g. synchronicity between multiple electrode recording sites and LED response sites), where the chest module control each of the arrays (e.g. specific focal epilepsy treatment algorithms).

Treatment is delivered through algorithms in the chest unit, which distill down to specific commands for the individual optrode sites (e.g. switch specific LEDs on for specified amounts of time and intensity; monitor/diagnose intended behaviour to ensure expected treatment is delivered, etc.), hence delivering the countermeasure to the focal seizure electric spike. For each optrode LED site, a crucial safety property is that LEDs cannot stay on for long, as this would cause intolerable temperature differentials, and consequently brain function impairments. Specifically, the regulatory rules state that the temperature rise should not exceed 20°C above body temperature. Other properties exist within optrodes, within the optrodes in the array, and within the array and the chest unit. The first optrode-specific CMOS chip (CANDO v3) we worked on and describe below were used for mice trials (2016). The current (CANDO v4) chip is being designed for use in primate trials (early 2020).

**Verification setup**

Once CANDO v3 was concluded, design documents, hardware design description files and device drivers existed. These were our starting point: the CMOS hardware architecture and instruction set (i.e. hardware commands and data layouts) and embedded device driver control software is written in C. The chip is fabricated in house, and the process automatically generates 115KLOC of constants, types, and other hardware-related structures. The user-written part of the device driver (3.1KLOC) is the actual C controller code we verified [29,30]. It comprises of three entities:

\(^{10}\) [http://www.cando.ac.uk](http://www.cando.ac.uk).
1. CMOS instruction set finite state machine (FSM);
2. Control system APIs, which assemble binary packets containing instructions and data for each optrode;
3. Control system main loop, which exercises the FSM and glues it together with serial communication.

As with the dialyser, it was not possible to start from design documents, so we had no choice but to start from the C code directly. We took the lessons learned from the dialyser and used VDM again as an intermediate language to help hunt for adequate formal specification before doing C verification using eCv.

The CMOS instruction set FSM model in VDM is simple (1KLOC) and captured a number of implicit invariants/assumptions not clearly stated anywhere else. It was written with (symbolic) executability in mind. That is, we can simulate the FSM behaviours and play with variations of test scenarios, otherwise possible only through error prone hardware/software debugging setups (e.g. debug device driver software controlling the CMOS hardware, which leads to unexpected outcomes if lag/delays are introduced due to debugging interrupting or interfering with execution within the hardware). Details of this work are in [29,30,35].

We specified types (e.g. 32-bit payloads, optrode struct, etc.), operations (e.g. switching LEDs on/off, diagnostics, etc.), and invariants over the FSM. In C, the optrode FSM controller is represented as a matrix of integers as bit vectors, from initial state and specific events to corresponding transitioned states. We represent the FSM as a series of partial maps (i.e. a map from Event to a map from State to State) with the following key invariants:

- VDM maps are total on known events and states; in C this is obvious because arrays are dense (i.e. allocated areas of an array entail all indexes are mapped to value);
- State to State (S) map (i.e. matrix columns) invariants were:
  - no state maps to the start (state);
  - start state can only be mapped to states {get_cmd, error};
  - error can only be mapped to states {get_cmd, error, chip_rst};
  - cmd_finish state can only be mapped to error;
  - get_cmd cannot map to any transmission state;
  - transmission states are the union of send and receive states;
  - send states are the states in set send_packet_x;
  - receive states are the states in set receive_packet_x;
- More involved invariants were encoded for the map of Event to map of State to State (E) map [29,30].

The overall (E) map has invariants, such as send states only communicate with receive states; any kind of chip reset must cause an error; all errors can only be recovered by restarting or resetting; only certain states can send, receive or create packets; and so on.

For the C code verification, we used the eCv tool (v.7.0.1) again. Because the CANDO device driver was in pure C, we chose eCv mostly because of MISRA-compliance checks, previous experience, its proof-audit traces, specification-suggestions for end-users and the fact that it participates in other certification processes for safety-critical systems. Other C verifiers would also work here, such as Microsoft’s SLAM or HAVOC.11

Results
The main outcomes from our analyses were:

1. VDM simulator of CANDO v3 CMOS instruction set FSM [29];

11 https://www.rise4fun.com.
2. Isabelle/HOL proof of correctness of the VDM specifications [30];
3. MISRA-C compliant device driver controller code [29];
4. Functional correctness verification of controller code [29].

After corrections in understanding, three serious scenarios were uncovered in the CMOS (CANDO v3) design. First, a key state related to programming the optrode memory was unreachable due to an earlier copy-paste error in the hardware design file, hence, all of its corresponding states were also unreachable. Second, packet data were being sent to a mistaken state due to a wiring problem. Finally, the chip reset command led to an unrecoverable state (i.e. FSM dead end). The first case was a coding mistake not observed during hardware testing, despite its potentially serious consequences in practice. The second case was a misunderstanding by the CMOS engineers and device driver programmers, which would entail the potential loss of a diagnostic signal, where consequences are unknown. The final case was known and simply an incomplete part of the design on what to happen under the specific reset conditions identified.

CMOS engineers valued the possibility of simulating the CMOS without all the complicated and error-prone instrumentation and low-level C details, as well as the precise documentation of their underlying assumptions. Device driver engineers appreciated the outcomes in terms of helping them identify issues, as well as ensuring that potential (error-prone) device driver encoding mistakes were caught as early as possible. Regulatory approval and trial administrators appreciated the emphasis on safety through mathematically precise documentation that enforced design decisions as they evolved and before critical primate and human trials started.

The VDM model has become a valuable resource for constructing future FSMs (CANDO v4), as well as maintaining the current one. Details about the VDM model are being adjusted/updated for CANDO v4, and the CANDO v3 model is part of the documentation used for certification.

The process of transforming the 1KLOC from VDM to Isabelle was manual at first, but we then created a tool in collaboration with the VDM developers from Aarhus, Denmark to automate the process [35]. The transformation of the VDM model to C was also novel: data refinement between VDM type representations (e.g. partial maps) and their corresponding C (e.g. bit vectors) have been proved using Isabelle/HOL. The C code templates corresponding to VDM were verified with eCv, as part of Newcastle computing projects [29,30].

The annotated and slightly modified code totalled 4.6KLOC, a not unusual increase from the original (3.1KLOC). The encoding process went through 8 different attempts, where the transformation to VDM was at the fourth attempt to encode contracts. After the first three attempts, we realised that too many low-level C details were hindering the finding of useful specifications. Arguably, VDM should have been used at first, yet it was not. This was due to the student involved being an experienced C programmer with an electronic engineering background, and being a bit reluctant to learn a new formal language, a common situation within engineering teams anyhow. Yet, the VCs started to get larger in number, repeated and difficult to discharge. For example, the packet to switch an LED on was written in C like this:

```c
(((Optrode0.Optrode_addr & BITS_6) << 18) | ((LED_ON & BITS_6) << 12) | (Optrode0.LED_addr & BITS_5)) &
BITS_24)
```

The number of masking (bitwise and $\&$) and casting (bitwise shift-left $\ll$) operations entailed various implicit type (bit size) widening and narrowing by the verifier, which entailed unnecessarily complex VCs (i.e. `Optrode_addr` is an unsigned `int` of 8 bits, of which only 6 are used, as per the mask; once shifted left by 18 bits, it becomes an unsigned `long` address/value). The final representation reduced the total VC number from 723 to 552 (a 24% reduction). There were 32 of such packet construction functions, where our approach to simplifying the way they were constructed and discharged considerably improved verification times. VCs were discharged in less than a minute, instead of 6–8 minutes.

Most of the VDM invariants discovered were upheld during the eCv verification for their C matrix representations, where some failed. Those that failed, we checked with stakeholders to ensure they were either typos, incomplete work, or anomalies/errors described above, rather than misunderstandings. This enabled us to fix errors and to influence how the next generation chip (CANDO v4) is designed in use for primate trials.

We demonstrated that the code is MISRA compliant and that all contracts for both packet assembly and control-interface commands are correct with respect to their specifications. We encode the VDM map as a matrix in C with column-view (e.g. first index on columns), which entails simpler embedding of most of the invariants. With the dialyser, we isolated hardware dependent libraries with minimal code contract axioms, so that we could use these (115KLOC) dependencies, yet did not verify them explicitly. Finally, following strategies in [54,55] that translates VDM to Isabelle, we used Isabelle/HOL to prove that the elicited contracts for the FSM are satisfied with respect to the overall FSM invariants. Thus, demonstrating that specifications are consistent: all operations are feasible (i.e. pre conditions implies post conditions), all invariants are sound (i.e. they are not empty/false), functions are applied within their domains, and so forth.

We are currently modelling the CANDO v4 CMOS chip instruction set architecture, as well as verifying the C device driver controlling the optrodes. These include the communication protocols between the software and the hardware devices. The work involves formal modelling and analysis, verification of implementation strategies and resulting products and deployment. It helps identify important potential flaws that were not detected during mice trials.

Socio-technical. The development team (mostly researchers/PhD students in electronic engineering) point blank refused any extra work/learning, given the intense milestone delivery timelines. It was through convincing the team leader about
the research potential (i.e. represent the FSM and the assessed risks mathematically in order to enable multiple kinds of analyses) that any work became possible at all. Eventually, some of the device driver programmers became curious, and in conjunction with their patience to explain various unclear/undocumented decisions, we managed to have them engage with the process. Once that happened, we could see a direct improvement in how the next stages of the device driver code got implemented: MISRA-compliance was easier to achieve, and the C code verification process was so much easier. Eventually the CMOS engineers also got onboard, and like our earlier friendly skeptics in the dialyser team, once they “saw” how mistakes could be prevented, engagement increased.

The CANDO v3 VDM model, Isabelle/HOL proofs and C device driver results are now part of the certification submission, as well as being used to inform the CANDO v4 design under way.

A.3. POLAR (early development-stage adoption)

Chronic disease can lead to end-stage organ failure, with transplantation as the only viable therapeutic option. Access to suitable donors is limited and despite best efforts, patients are placed onto wait-lists, where they may become too sick or even die before receiving an opportunity for life-saving transplant. This process frequently involves surgical procurement of donor organs from deceased donors at geographically disparate hospitals and rapid transport to implantation centres at odd hours. Organs are typically cooled down by flushing with specialised organ preservation solutions in an effort to reduce metabolism and facilitate transport. Unfortunately, organs have a limited tolerance to these periods without blood-flow (termed ischaemia), hence introducing logistical concerns and necessitating a rushed process.

These challenges have led to the investigation of a variety of organ preservation technologies aimed at preserving organs’ function following removal from a donor [56]. A key challenge limiting implementation of such technologies is the necessity for costly specialist personnel monitoring of devices within an already financially stretched system. Here, we describe efforts to safely automate such a device, towards eliminating this requirement and facilitating clinical application.

System

The POLAR system consists of a closed-loop control maintaining organ function. The system control algorithm is responsible for keeping the organ alive, as well as keeping an audit trail of what happened when. A series of alarms and their target users (i.e. maintenance engineers, retrieval team, recipient team, transport team, etc.) are triggered under specific physiological conditions.

Initial human research organ testing has successfully taken place (August 2019), where expected machine behaviours were maintained on a live human organ for over 24 hours, exceeding our stated target goal of 24 hours. The next step will be to assess the device’s functional and clinical efficacy in preserving organs as it moves into clinical trials. Due to commercial sensitivity and pending patent applications, we cannot give further details.

Verification setup

The verification setup for POLAR is still being investigated. So far, requirements have been described using problem frames [31], the control algorithm has been defined in VDM, and the circuit design itself has just been tested on live human organs.

We started work on eCv for to the verification of the embedded device driver and circuit board controller C and C# code, respectively. We are also investigating the use of a new theory of summaries being developed to enable simplification and interpretation of logged data accurately and timely (i.e. accurate with respect what instruments’ readings were, yet timely so that a correct summary can be given to the transplant surgeon). The device driver code is 5.1KLOC of C and 500LOC of assembler.

Results

Different from the previous two examples, where formal verification interventions occurred at the end and in the middle of development, formal verification is part of development from the beginning in the organ preservation control system example (since 2018). This enabled an understanding of the challenges presented, influenced the requirements [31] for such a device and facilitated the creation of bespoke algorithms building on existing theories in formal methods.

These efforts are resulting in improved functionalities for the underlying control software and impacted the physical design process. This device is still under development, has commercial interests and will have to be certified. Due to this and ongoing IP considerations for algorithms and initial formal control-system designs, further details on implementation will appear in future publications.

Socio-technical. Given the early application of formalisms during initial designs, no post-hoc verification took place. This was possible thanks to the POLAR engineering team-lead’s belief that the application of formalism can and will deliver better outcomes. Interestingly, the team lead is a biomedical engineer with a background in medicine; a stakeholder with different background to the other two examples above.

Device design and software development teams typically sit detached from the regulatory and clinical environments that their devices and programs operate from. Our experience to date has been that the closer the project leads sit to the highly
regulated medical environment, the faster they have been to “buy in” to the potential benefits that formal methods present to their devices.

To address this socio-technical issue, and pave the way for adoption on a larger scale, it will be important to reach out to the medical community and explain the benefits that formal methods offer to clinicians and their patient groups. Once these formal processes are understood, we anticipate that patients and clinicians will demand the application of such principles as novel devices are developed, ruling out certain classes of errors by design, hopefully ultimately influencing how certification takes place in future.

References

[1] E. Topol, The Topol Review: preparing the healthcare workforce to deliver the digital future, Tech. Rep., Health Education England, February 2019.
[2] P.S. Authority, Baxter colleague volumetric infusion pump: product recall notification, Patient Saf. Advis. 4 (1) (March 2007).
[3] H. Thimbleby, Improving safety in medical devices and systems, in: IEEE International Conference on Healthcare Informatics, 2013.
[4] J. Van Den Bos, K. Rustagi, T. Gray, M. Halford, E. Ziemkiewicz, The $17.1 billion problem: the annual cost of measurable medical errors, Health Aff (Millwood) 30 (4) (2011) 596–603.
[5] D. Vogel, Medical Device Software Verification, Validation, and Compliance, Artech House, 2010.
[6] A. Osaiweran, M. Boosten, M. Mousavi, Analytical software design: introduction and industrial experience report, Tech. rep., Philips Medical and Verum Technologies, 2010.
[7] M.D. Harrison, L. Freitas, M. Drinnan, J.C. Campos, P. Masci, C. di Maria, M. Whitaker, Formal techniques in the safety analysis of software components of a new dialysis machine, Sci. Comput. Program. 175 (2019) 17–34.
[8] J.C. Campos, G. Doherty, M.D. Harrison, Analysing interactive devices based on information resource constraints, Int. J. Hum.-Comput. Stud. 72 (3) (2014) 284–297, https://doi.org/10.1016/j.ijhcs.2013.10.005.
[9] L. Lin, R. Isla, K. Doniz, H. Harkness, K.J. Vicente, D.J. Doyle, Applying human factors to the design of medical equipment: patient-controlled analgesia, J. Clin. Monit. Comput. 14 (4) (1998) 253–263.
[10] Ayse P. Gurses, Peter Doyle, Medical devices in the “wild”, Patient Safety Newwork, December 2014.
[11] J.C. Campos, A structured approach for investigating the causes of medical device adverse events, J. Med. Eng. 2014 (2014) 13, https://doi.org/10.1155/2014/314138.
[12] The global medical device market (https://markets.businessinsider.com/news/stocks/the-global-medical-device-market-is-expected-to-reach-an-estimated-409-5-billion-by-2023-and-it-is-forecast-to-grow-at-a-cagr-of-4-5-from-2018-to-2023-1027478175) [online], 2019.
[13] U.D. of Health, H. Services, General principles of software validation; final guidance for industry and FDA staff, Tech. Rep. UCM085281, Food and Drug Administration, Jan 2002.
[14] BSI. Medical device software — software life-cycle processes, Tech. Rep. BS EN 62304, British Standards (BSI), May 2011.
[15] S. Nordhoff, Do-178C/ed-12C: the new software standard for the avionic industry: goals, changes and challenges, Tech. Rep., Software Quality Systems, June 2011.
[16] Food and Drug Administration (FDA), Class 2 Device Recall ACCUHEK Connect Diabetes Management App, https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfRES/res.cfm?id=134687, 2015.
[17] D. Copper, J. Barnes, Tokeneer ID station EAL5 demonstrator: summary report, Tech. Rep. SP1229.811 Issue: 1 Altan-Praxius, August 2008.
[18] J. Woodcock, P.G. Larsen, J. Bicarregui, J. Fitzgerald, Formal methods: practice and experience, ACM Comput. Surv. 41 (4) (Oct 2009).
[19] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, S. Winwood, sel4: formal verification of an operating-system kernel, Commun. ACM 53 (6) (2010) 107–115.
[20] MHRRA, Rules and Guidance for Pharmaceutical Manufacturers and Distributors, Pharmaceutical Press, 2017.
[21] S. Bonfanti, A. Gargantini, A. Mashkoor, A systematic literature review of the use of formal methods in medical software systems, J. Softw. Evol. Process 20 (e1943) (2018) 1–18.
[22] A. Osaiweran, M. Schuts, J. Hooman, Experiences with incorporating formal techniques into industrial practice, Empir. Softw. Eng. 19 (4) (2014) 1169–1194, https://doi.org/10.1007/s10664-013-9251-2.
[23] P.J. Hopcroft, G.H. Broadfoot, Combining the box structure method development and CSP for software development, Electron. Notes Theor. Comput. Sci. 128 (6) (2005) 127–144, https://doi.org/10.1016/j.entcs.2005.04.006.
[24] T.S. Hong, C. Sook, A. Salehi, M. Butler, L. Ladenberger, Validating and verifying the requirements and design of a haemodialysis machine using the Rodin toolset, Sci. Comput. Program. 158 (2018) 122–147, https://doi.org/10.1016/j.scico.2017.11.002.
[25] P. Arcaini, S. Bonfanti, A. Gargantini, A. Mashkoor, E. Riccobene, Integrating formal methods into medical software development: the ASM approach, Sci. Comput. Program. 158 (2018) 148–167.
[26] H.D. Macedo, P.G. Larsen, J. Fitzgerald, Incremental development of a distributed real-time model of a cardiac pacing system using VDM, in: International Symposium on Formal Methods - FM, in: Lecture Notes in Computer Science (LNCS), vol. 5014, Springer, 2008, pp. 181–197.
[27] P. O’Hearn, Continuous reasoning: scaling the impact of formal methods, in: 33rd Annual ACM/IEEE Symposium, ACM, 2018.
[28] B. Cook, Formal reasoning about the security of Amazon web services, in: H. Chockler, G. Weissenbacher (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2018, pp. 38–47.
[29] A. Pollitt, Verifying the CANDO Project Optredo Command Interface in eC, Master’s thesis, School of Computing Science, Newcastle University UK, August 2018.
[30] B. Wooding, Using Formal Methods and Proof to Verify a CANDO Epilepsy Medical Device, Master’s thesis, School of Computing Science, Newcastle University UK, June 2019.
[31] K. Marco, Verification of the POLAR Control Interface, Master’s thesis, School of Computing Science, Newcastle University UK, August 2018.
[32] V. Kassian, Methodologies for high-assurance medical devices, Master’s thesis, School of Computing Science, Newcastle University, Oct 2014.
[33] J. Ginty, Approach to Writing Functionally Correct Code for use in Medical Equipment, Master’s thesis, School of Computing Science, Newcastle University UK, Aug 2015.
[34] E. Martinkute, Formal Specification of a Novel Infant Dialysysr, Master’s thesis, School of Computing Science, Newcastle University, UK, Aug 2016.
[35] J. Simm, Creating a Tool for Translating VDM to Isabelle/HOL, Master’s thesis, School of Computing Science, Newcastle University UK, July 2019.
[36] J. Carlson, D. Crocker, Escher Verification Studio Perfect Developer and Escher C Verifier, J. Wiley and Sons, 2012, pp. 155–193, Ch. 5.
[37] P. Nair, An experiment on program development, BIT 12 (1972) 347–365.
[38] MISRA Ltd. MISRA-C:2012 Guidelines for the use of the C language in critical systems, Tech. Rep. MISRA:2012, MISRA Ltd, March 2013.
[39] G. Travis, How the Boeing 737 Max disaster looks to a software developer, IEEE Spectrum, April 2019.
[40] P. Masci, Y. Zhang, P. Jones, H. Thimbleby, P. Curzon, A generic user interface architecture for analyzing use hazards in infusion pump software, in: V. Turau, M. Kwiatkowska, R. Mangharam, C. Weyer (Eds.), 5th Workshop on Medical Cyber-Physical Systems, in: OpenAccess Series in Informatics (OASIcs), vol. 36, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2014, pp. 1–14, http://drops.dagstuhl.de/opus/volltexte/2014/4518.
[41] M. Harrison, J. Campos, P. Masci, Patterns and templates for automated verification of user interface software design in PVS, Tech. Rep. TR-1485, School of Computing Science, Newcastle University, 2015.

[42] M.G. Couturier, J. Crosier, C. Griffiths, J. Smith, M. Drinnan, M. Whitaker, R. Beckwith, J.N.S. Matthews, P. Flecknell, H.J. Lambert, Haemodialysing babies weighing < 8 kg with the Newcastle infant dialysis and ultrafiltration system (NIDUS): comparison with peritoneal and conventional haemodialysis, Pediatr. Nephrol. 29 (10) (2014) 1873–1881, https://doi.org/10.1007/s00467-014-2923-3.

[43] MISRA Ltd. MISRA-C++:2008 Guidelines for the use of the C language in critical systems, Tech. Rep. MISRA C++:2008, MISRA Ltd, June 2008.

[44] L. Martin, Joint strike fighter air vehicle C++ coding standards for the system development and demonstration program, Tech. Rep. 2RDU00001 Rev C, Lockheed Martin, December 2005.

[45] D. Crocker, Can C++ be made as safe as SPARK?, Ada Letters 34 (3) (2014) 5–12, https://doi.org/10.1145/2692956.2663179.

[46] A. Hevner, H. Mills, Box-structured methods for systems development with objects, IBM Syst. J. 32 (2) (1993) 232–251, https://doi.org/10.1147/sj.322.0212.

[47] World-Health-Organization, Epilepsy, www.who.int/mediacentre/factsheets/fs999/en/, Feb 2018. (Accessed 8 April 2018).

[48] E. Iseri, D. Kuzum, Implantable optoelectronic probes for in vivo optogenetics, J. Neural Eng. 14 (3) (2017) 031001.

[49] P. Hegemann, G. Nagel, From channelrhodopsins to optogenetics, EMBO Molecular Medicine 5 (2) (2013) 173–176.

[50] F. Mazda (Ed.), Telecommunications Engineer’s Reference Book, Butterworth-Heinemann, 1993.

[51] H. Zhao, A. Soltan, P. Maaskant, N. Dong, X. Sun, P. Degenaar, A scalable optoelectronic neural probe architecture with self-diagnostic capability, IEEE Trans. Circuits Syst. I, Regul. Pap. 65 (8) (2018) 2431–2442.

[52] F. Dehkhoda, A. Soltan, R. Ramezani, H. Zhao, Y. Liu, T. Constandinou, P. Degenaar, Smart optrode for neural stimulation and sensing, in: SENSORS, 2015 IEEE, IEEE, 2015, pp. 1–4.

[53] H. Zhao, F. Dehkhoda, R. Ramezani, D. Sokolov, P. Degenaar, Y. Liu, T. Constandinou, A CMOS-based neural implantable optrode for optogenetic stimulation and electrical recording, in: Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE, IEEE, 2015, pp. 1–4.

[54] L. Freitas, I. Whiteside, Proof patterns for formal methods, in: Formal Methods, Formal Methods Europe, Springer, 2014.

[55] L. Freitas, C.B. Jones, A. Velykis, I. Whiteside, How to say why, Tech. Rep. CS-TR-1398, Newcastle University, November 2013, www.ai4fm.org/tr.

[56] S. O’Neill, et al., Novel organ perfusion and preservation strategies in transplantation – where are we going in the UK?, Transplantation (2020), https://doi.org/10.1097/TP.0000000000003106.