Informal control code logic

Bergstra, J.A.

Publication date
2010

Document Version
Submitted manuscript

Link to publication

Citation for published version (APA):
Bergstra, J. A. (2010). *Informal control code logic*. arXiv.org. http://arxiv.org/abs/1009.2902

Informal Control Code Logic

Jan A. Bergstra[*]

University of Amsterdam, Faculty of Science, Informatics Institute,
Section Theory of Computer Science,
Science Park 904, 1098 XH, The Netherlands[†]

September 12, 2010

Abstract

General definitions as well as rules of reasoning regarding control code production, distribution, deployment, and usage are described. The role of testing, trust, confidence and risk analysis is considered. A rationale for control code testing is sought and found, for the case of safety critical embedded control code.

Keywords: (polyadic) control code, box, execution, testing, risk analysis.

1 Introduction

Although the combination of hardware and software into functional units pervades computing technology the question why this organizing principle of information technology is so powerful and why it has emerged during the evolution of digital equipment is intriguing, if only from a theoretical viewpoint. Following the terminology of [13] software is understood as control code and data code. In this paper a machine is called a box. A control code logic (CCL) provides formal reasoning methods that apply to control code and code controlled boxes. According to [13] conditional equations constitute a workable formalism for a CCL.[1] Informal control code logic (ICCL)[2] provides informal reasoning patterns about the use of control codes.

Many ways to approach this kind of question can be imagined. The history of technology can be followed in detail, conjectural histories may be developed (quite common in the theory of money, see [7]), or economic models may be construed from which the advantages and disadvantages of technological strategies can be analyzed.

None of these approaches are available to us at this stage because of the lack of a general conceptual model of the situation. The primary objective of this paper is to develop that conceptual model. It seems to be the case that given such an objective one needs to proceed from scratch. I will proceed in a holistic way by discussing a number of aspects:

[*] This work has been carried out in the context of NWO Jacquard project Symbiosis.

[†] P.O. Box 94214, 1090 GE Amsterdam, The Netherlands E-mail: `j.a.bergstra@uva.nl`

[1] A brief summary of [13]: (i) control codes are either produced via instruction sequences by means of compilation or differently, for instance machine learning (such control codes are termed dark programs in [25]), (ii) no substantial definition of 'executable code' can be found, rather executability needs to be introduced as a primitive predicate together with a machine model, (iii) some basic facts about compilers and assemblers can be phrased in terms of conditional equations (in particular several forms of fixed points), (iii) a formalization of some arguments from [4], [22] and [19] (iv) fairly detailed definitions of installation and portability are given in terms of CCL only.

[2] In [24] another occurrence of a specialized informal logic is presented, and [23] discusses (fallacious) reasoning rules in the same field.

1. I will make use of imaginative definitions as presented in [7], and in particular the notions of an IDBR (informal description by role), an LSCD (logical solitary concept definition) and an SCFD (stratified concept family definition). An IDBR for a code controlled device (also called a box) is provided in the line of the machine function based definitions of [13].

2. I will make use of synthetic execution architectures as defined in [14] and of a generalized form of the analytic execution of [14].

3. Boxes are instances of box types. Box types develop along a path of technological evolution. Within a type a sequence of versions (subtypes) may be distinguished. Control codes can be exchanged between boxes and information carriers. A classification of exchange mechanisms is proposed.

4. A variety of control related actions are brought into the picture:

   - shipping control code, (thrashing control code),
   - loading control code, (unloading control code),
   - decision to use a box, (decision not to use a box),
   - usage (of a box) = carrying out the decision to use a box
   - idling (of a box) = carrying out the decision not to use a box
   - utility (when making successful and intended use of a box),
   - disutility (when making use of a box without intended success),
   - expected utility, (ex ante assessment),
   - (expected disutility),
   - satisfycing usage, (dissatisfycing usage),
   - [user side/producer side] control code testing,
   - producer side control code validation,
   - producer side control code verification.
   - user side control code deployment risk analysis
   - producer side control code release risk analyisis

5. The primary category to which all other actions of agents, including users and producers, is that of decision making. Testing, risk analysis, reputation assessment, confidence awareness, and trust maintenance all contribute to decision making. It is assumed that under normal conditions the use of a control code does not take place without a preceding decision to to so.

Each of these aspects plays a role in the construction of an ICCL level of abstraction for describing computer technology. Perhaps too many aspects have been collected below and a more concise discussion can be developed. In preparation of the discussion I mention two philosophical perspectives that somehow threaten and potentially overshadow my project in each of it stages.

1.0.1 Looking for meanings is futile

I am tempted to ask questions like: “what is a control code,’, “what is a code controlled device”, “what is testing a control code”, “what is a computer” and so on. Another style of phrasing would be: ‘what is the meaning of “control code” ’, ‘what is the meaning of “code controlled device” ’, ‘what is the meaning of “testing a control code” ’, and ‘what is the meaning of “computer” ’. Now in [1] Alston expands in long detail on the impossibility of finding such meanings. He considers the search for meanings futile because there is no general class of “meanings” within which one is searching. As a consequence I will choose this strategy: instead ask: 'what is a definition of "control code" ', 'what is a definition of "code controlled device" ', 'what is a definition of "testing a control code" ', and 'what is a definition of "computer" '. And in order to escape from Alston's critique a meaning needs to be assigned to this use of the term "definition". That is done by making use of a classification of definitions taken from [7]. Nevertheless it is easy to ask for meanings of seemingly straightforward terms and phrases and to underestimate the force of Alston's critique. Alston formulates a difficulty that each quest for meaning needs to confront.

When writing about such meanings work of previous authors needs to be used. Years ago Jervis [26] formulated the following criticism (then concerning some books he was reviewing): "Although there are lots of footnotes and references, in most cases these are the obligatory nods that substitute for refining and building on what others have done." Avoiding this problem is a very difficult objective indeed and I am very reluctant to claim successfully having done so.

1.0.2 NOMA

As I will discuss in more detail below testing can produce knowledge about control code and in particular about the behavior of a code controlled box upon its execution. This knowledge may consists of reproducible data about physical phenomena. Quite at the opposite end of the spectrum one may envisage a control code which has been obtained via compilation of an instruction sequence which is supposed to be executed by a box for which a formal model of execution (in terms of the underlying instruction sequences) is given. Then predictions about behavior can be made with mathematical (logical) precision.

Unclear is to what extent both forms of knowledge can coincide or overlap. It is almost inconceivable that pure formal reasoning can produce a valid prediction about the behavior of a physical device. Somewhere the outcome of formal reasoning must be combined with experimental data to arrive at a prediction of observable quantities. Thus experimental data (obtained via control code testing) and logically derived information seem to exist in different worlds.

Whether or not these worlds actually differ is left open below. I will in some cases assume that there may be an alternative to testing for obtaining certain information. Whoever thinks that testing is insufficiently reliable for control code validation must believe that such alternative methods for obtaining information can be found, at least under certain conditions. Here lies a philosophical issue that will not be further analyzed below, in spite of the fact that its resolution seems to be rather essential for our objectives.

If one concedes that both methods of knowledge acquisition lead to disjoint sets of possible results one may wonder whether or not this form of seemingly related but in fact non-overlapping methods of investigation constitutes new circumstances in terms of the philosophy of science. Such is not the case, however. Following [32] an example of non-overlapping theories is found when contemplating the results of science in contrast with the results of theological reflection. Moritz, in [32] discusses NOMA: non-overlapping magisteria of authority (NOMA), a concept put forward by the ethologist S. J. Gould. Moritz concludes, however, that NOMA inadequately describes the relation between science and theology because according to him overlaps between these areas cannot be excluded.

An asymmetry between positive and negative information may be at hand, however. In the case of control codes, confidence generating information obtained from tests may be disjoint from confidence generating information obtained by means of logical analysis while logical analysis might lead to a confidence loss (negative information) which subsequently suggests useful tests (the outcome of which may confirm the same negative information). This state of affairs is comparable to the fairly detailed NOMA related viewpoint put forward by Stenmark in [38]. Making a connection between ICCL and this perhaps marginal part of philosophy of science may seem quite farfetched. But the presence of some form of NOMA, though regarding quite different areas of authority, is not entirely farfetched. The huge gap between those who insist on software testing and those who insist on formal software verification, both groups of individuals often not trusting one-another's methods, is a stable and significant phenomenon which calls for a further explanation beyond the suggestion that the methodology of formal verification is still largely unfamiliar to most who have been used to testing.

1.1 The ICCL level of abstraction

If computers and computer systems are considered from an ICCL perspective that means first and for all that no hypothesis is made and no understanding is assumed in any form concerning the mechanical aspects of control code execution.[3]

To what extent an ICCL perspective on computing can be maintained is unclear. Further development of ICCL may extend the frontiers of ICCL and reveal its intrinsic boundaries so to speak. It is considered to be consistent with ICCL to make assumptions about the bookkeeping and management of bit sequences within a system. Below that will hardly be done, but explaining the operating architecture of a system in terms of a number of self-explanatory file management primitives (commands) is needed for an investigation of (i) the simultaneous presence of control codes for different applications within the same system, and (ii) of the distinction between control code ad data code as well as (iii) the way data code is preserved between sessions of execution of the same control code.

Instruction sequences can be used to specify threads for issuing commands to an operating architecture. In [13] a proposal for an operating architecture is formulated. Threads are specified that embody definitions of control code installation and control code portability. Similar specifications can be found in [16].

A major incentive for this work has been to develop some theory about the business cases for software process outsourcing. That led to the objective to disentangle that issue entirely from the endless complexities of control code execution mechanics and control code structure. This application perspective is not pursued below and from the standpoint of software process sourcing (or software asset sourcing as it has been named alternatively) only preparations are made. It seems clear, however, that significant progress concerning operating architecture description is needed if significant conclusions regarding in- and out-sourcing decisions are to be drawn from consideration of the matter at the abstraction level of ICCL.

1.2 Setting the scene

The term satisficing usage is coined following the terminology of H.A. Simon for decisions in the context of decision making. Having available this terminology we provide a number of preparatory remarks.

1. We may assume that a box B loaded with control code C creates for its user U (when used at its initiative) either a utility (UT) or a disutility (dUT). The only way that UT, provided it is non-trivial (that is nonzero), can come about given that C and B are both at hand for U is by U's executing C on B. The same holds for dUT if it is nonzero. It is the effect of a failed execution which is of a lesser utility than not to execute C on B at all.[4] Here it is assumed that no other comparable boxes and codes are ready for use by U as an alternative.[5]

2. Usage (of a box, or of a control code for a box) involves the concept of execution of a box. This implies that when writing about it a box on which the usage is performed needs to be

[3] In particular the common explanation that this is a matter of computer program execution is rejected as being uninformative.

[4] Not executing C on B may deliver some disutilities as well but it can be done without having C or B or both C and B at hand. But an unsuccessful execution of C on B may create a disutility which is far more problematic than not executing C at all if during the execution of C before a failure took place a new state has been formed from which recovery is only possible by making use of the tail of an execution of C on B.

[5] Thus for instance mere code inspection of C by U cannot bring forward either the mentioned utility or the mentioned disutility because it fails to involve an execution of the code.

so ‘real’ that it can be executed. But of course it isn’t. So here is a key difficulty for writing about usage: one intends to write about effects only achieved by execution while writing about abstractions that cannot possibly execute.

3. Usage is a concept with several close relatives: for instance experiment, demonstration and test (including validation). Except interpretation and simulation each of these variations on the theme of usage involves an execution of the box. But the test provides none of the utilities that usage provides and instead it provides quite different utilities (if at all). A test cannot deliver a disutility (though a false positive comes close). The following variations of usage can be distinguished.

   Satisficing usage. Practical activity performed by one or more users (sometimes hierarchically stratified in supporting users and end users al of whom are considered to be human beings) centered on one or more runs of the box, performed, to U’s satisfaction, to meet U’s objectives, as embodied in the utilities of usage.

   Usage. Practical activity essentially involving one or more runs of the box, performed to meet U’s objectives, as embodied in the utilities of usage (and with failure captured by the disutilities of usage).

   Interpretation. An interpretation is a simulation which is performed by way of usage. So it may deliver utilities close to UT or disutilities close to dUT.

   Demonstration. Demonstration also involves box runs and its outcome is knowledge for the user or its representatives.

   Test. Practical activity involving one or more runs of the box, that will not deliver any of the utilities or disutilities of usage, but instead it will provide knowledge about the box. This knowledge is in principle to be used for decisions about usage or about shipping of boxes and/or control code.

   Experiment. An experiment involves runs of the box. Its outcome is a hypothesis about box behavior. Such hypothesis may be further examined by way of test. Inductive inference leads from experiment to hypothesis.

   Simulation. A simulation does not involve a run of the box. It involves run’s of a modified control code on a modified box (at least one of the modification is non-trivial or else the simulation is just an experiment.) No usage utilities (disutilities) will result. The objective of a simulation is hypothesis formation (concerning the target box) to be further reinforced by means of experiments and thereafter examined by way test.

4. The viewpoint that tests require a hypothesis for confirmation or for disconfirmation while experiments may generate a hypothesis is incongruent with the experimental sciences where the above ‘test’ is rather termed an experiment.

There is a plausible order of events as follows: once a control code has been produced its simulation on a machine different from a target machine may generate hypotheses about its functionality. Subsequently an experiment on the target boxes may reinforce these hypotheses, subsequently the hypotheses are subject to tests. If passed demonstrations are due, followed by usage and upon a positive evaluation of the usage a phase of satisfycing usage may follow.

1.3 Control code exchange

The crux of code controlled boxes is that control codes contained in a box can be exchanged. “Box with exchangeable control code” is another phrase for “code controlled box”.[6] For box users the

[6] Alternatively the feature of a box that it admits exchangeable control code is termed the exchangeable control code feature, or simply the control code feature.

advantage of exchangeable control code must be connected to their performing the act of control code exchange at least in some cases. More precisely a user obtains an advantage from the mere possibility of control code exchange by being certain that it can be done even if that is so rare that most users never make use of a control code exchange. The minimal role of the control code feature is to serve as an insurance policy for a box owner that his box can be made useful for other purposes when needed.

This implies that in order to assess the feature of exchangeable control code for boxes some clarity about the exchange mechanisms must be provided. It is taken for granted that exchanging control code can be viewed as a combination of dropping a loaded control code which is considered unproblematic and loading another control code. As it turns out several scenarios exist for loading new control codes into a box. Here are some options:[7]

Plugged external memory device (PEMD). This refers to a CD (in former day a floppy disk), or a memory stick, which is connected directly to the box without any network in between. Controls on the device allow downloading data (bit sequences often called files) from the PEMD and uploading data from the box to the PEMD.

Wired EMD. An external disk drive is an example: the EMD has some functionality usually requiring independent power supply; once a cable connection has been established between EMD and box exchanges can be controlled from the box's user interface.

Wireless EMD. Connection with an EMD may be established via bluetooth or a comparable technology which does not presuppose full internet connectivity.

Wired internet connection. A statically positioned box may have IP capability and be internet connected via a wire.

Wireless internet connection. This comes in several flavors.

Very local area network. At the smallest scale a bluetooth connection may connect the box to an internet host from a small distance (several meters).

Local area network. At a small scale (say up to 20 meters) a network may provide wireless link to a local host, without options for roaming.

Wide area network. At a larger scale a system of base stations may provide connectivity from the box throughout a wider area, say a campus comprising some square kilometers.

Global network. World wide connectivity may be provided if the box contains a mobile phone.

Loading a new or improved CC requires conscious user activity if a PEMD is used, and it may also require conscious action activity if wired or wireless EMD is used. All methods from Wired internet connection onwards are said to be internet based.

1.3.1 Network based exchange implies exchangeable control code.

Remarkably the issue of determination the rationale (or different rationales) of exchangeable control code as a feature of box technology is far from independent from these different options for control code exchange. If PEMD is used (for instance a CD disk, or historically a floppy disk) it is certainly conceivable that a box performs tasks that may as well be performed by boxes of different design that are not code controlled. But if a box has global wireless internet connectivity via the mobile phone network, it is most plausible to assume that its processing capabilities are so advanced

[7] Besides ranging from technologically simple to quite intricate, this listing corresponds with a historical development in technology as well. This paper not being about history of computing it may be safer to attribute the listing some virtue concerning the chronology of a conjectural history, perhaps one may even speak of evidence based conjectural history.

that it must be code controlled and if it is not, that option has been blocked in hindsight after a satisfactory code controlled prototype had been constructed during the design and development of the box.

Assuming that machines are more likely to be code controlled with growing functionality[8] it appears that the question why code controllability is so important is mainly meaningful if simple methods of CC exchange are considered such as PEMD, wired EMD and wireless EMD. Considering current and past technology one may agree that wireless global area network connectivity is present only for boxes for which the control code feature is plausible which for that reason is available by default, and for which its absence rather than its presence is expected to be specifically engineered, for instance for security reasons.

1.3.2 Hidden versus visible control code exchange.

The observation that the control code feature's plausibility depends on assumptions about code exchange mechanisms leads to the proposition that exchangeable control code comes in different flavors:

visible control code feature: control code exchange is only performed as an immediate consequence of conscious user activities. Such activities are classified as configuration management by hand, or equivalently as interactive configuration management.

hidden control code feature: changes made only by autonomous system actions either initiated from an external network component or from the box at hand but in both cases hidden from a user. This is the hidden control code feature, it involves automatic configuration management. Of course if many control codes are simultaneously loaded a mixture of automated and interactive configuration management can be in charge for control code exchange on the same box.

illegal control code feature: if control code exchange can be achieved and in addition it can only be achieved by a conscious but illegal or unintended (from the perspective of the box provider) interactive configuration management by the user the box offers an illegal control code feature. If control code has been exchanged under these conditions the box is said to have been hacked. Of course mixtures with the visible CC feature and the hidden CC feature can be imagined.

illegal hidden control code feature: this applies if automated activities can lead to control code exchanges that are illegal or unwanted from the perspective of the box provider (and in most cases also from the perspective of the box user). Viruses and worms, Trojan horses and more general malicious control code (malcode in brief) usually enter a box in this fashion.[9]

1.3.3 Conceptual questions about the control code feature.

The question to explain the large number of code controlled devices seems to split into different parts:

1. Why is it the case that beyond a certain level of complexity and functionality all boxes are code controlled?

2. What explains the presence or absence of the code control feature for box types with functionalities that do not by their own complexity indicate the code control feature as the most

[8] This assumption itself reflects limitations. Human brains are complex but seem not to be code controlled.

[9] In some cases a control code exchange that imports a malicious code will require user activity while the user is unaware that this is a consequence of his actions. Then we may speak of a user mediated hidden control code feature. It is implicit that the user is unaware of the mediating role of his actions which most likely are performed with another purpose, with opening an email attachment as a paradigmatic example.

plausible default? (For instance: why do digital photo cameras not offer the control code feature?)

3. Concerning the concept of computer science and its methodology several questions emerge:
    - is a computer a box with code control feature by default or is that merely and option?
    - If a computer is by default code controlled is it supposed to be universal in some sense?
    - If computers are considered not to be code controlled (and for that reason not to be universal either), what is the characteristic subject of computer science which provides its conceptual identity?
    - If the latter questions have unsatisfactory answers due to the lacking expressive power of CCL: can computer science more convincingly be defined as the science of tasks that can be performed by code controlled boxes with codes being produced by compiling instruction sequences.
4. How to express in terms of code controlled boxes that a control code is malicious? How to express (if that can be done) that a malicious control code is a virus, a worm, a Trojan horse and whatever other kind of malcode has been distinguished?

2 Control code related decision making

I will assume that all actions involving control code are performed by conscious agents and that each action is preceded by a decision tot that extent. The decision to use a specific control code can be an involuntary one in the sense of [21], for instance if malicious code is being activated. That case will not be dealt with below. Decision making concerning control code can vary from a slow and strategic process involving many agents and moving through a pattern of formalized steps as if a sequential program were executed (following H.A. Simon's description of decision making, see [36, 18], and referred to as formality of a decision in for instance [5]) to a real time action disallowing much opportunity for systematic reflection. A typical study on real time decision making is [29] analyzing the decision taking behavior of traffic participants in front of traffic lights. This may be compared to internet users who may decide about control code activation each time they make up their mind about opening an email attachment. In this paper I will assume a highly informed decision making process, thus avoiding the complications of empirical investigation of actual decision making processes hampered by real time pressure and a correlated lack of information. This assumption is by no means realistic but it creates more idealized circumstances that may simplify the theoretical work.

2.1 Making the decision to use a specific control code

Before U puts a control code into use by having it executed on box B for some specific purpose U will make a decision to proceed in that way. Devoid of cognitive insight in the structure of the control code he will need another basis for making the decision.

The user U of a control code primarily depends on his trust in the control code producer. This trust may be based on any combination of the following matters:

- Past experience with other control code products manufactured by the same producer.
- Knowledge about the production process that has been applied. This knowledge can be based upon reputation analysis or upon detailed production process inspections performed by U or on behalf of U.
- Reputation of the control code as a product obtained by analyzing the satisfaction of previous users.

Besides acquiring confidence that the control code is adequate based on trust in its producer users can perform tests and users can obtain information about other instances of the same the control codes from other users they happen to know and in whom they have sufficient trust.

Besides reputation based trust of its producer and trust based upon production process inspection testing is the only method available to a user to increase his confidence that the control codes will lead to satisfycing usage. This dominant role (and interaction) of reputation, trust, test and confidence is specific for a setting where ICCL is leading.[10]

So it can be concluded that when it comes to decision making ICCL assigns a central role to control code testing, and because of this central role testing is investigated next in more detail. The general question that needs to be answered is this: what knowledge about C can be generated by running B(C) with a number of different inputs. Control code testing takes place at least at two sides: user and manufacturer. Intermediate agents like software salesmen, system administrators, IT consultants may perform tests as well, but these agents (or rather agent roles) will be ignored in the sequel of this paper.

2.2 Decision making in general

Scanning the literature on decision making I get the impression that the foundational question "what is a decision" has not attracted much attention.[11] Yet for the current objectives it is a rather intriguing question to settle that matter in more detail. Below I will put forward an elementary theory of decisions. It is not supposed to have any empirical content. I assume that an agent or a collection of agents proceeds by performing actions of various kinds. These actions include observations of atomic propositions and all actions may have side-effects. A classification of actions has been made into operational actions henceforth called primary actions which directly constitute to the major performance of these agents and collateral actions henceforth called secondary actions which involve planning, communication, what is usually called decision making and so on.

When considering a trace of a system its secondary actions can be grouped together in threads of actions that are semantically connected. The secondary actions take place as if a multi-threaded system proceeds through one of its strategic interleavings. Each of its threads may be understood as the execution of some sequential process (or program in the terminology of [36]). Focusing on one of the threads involved an external observer may or may not be justified in making the claim that this thread constitutes (represents or documents) the process of making some decision. It is a decision if that claim is valid. Several grounds for validity of such claims can exist. Quite common is the situation that many of its actions are labeled with comments about their being constituents of decision making process. For a "formal theory" of decision (formal decision process view) that criterion will suffice: a decision is a tread of actions which conforms one of the possible alternatives of a pre-existing grammar for decision making processes. This grammar will include label assignments of various actions that explain which particular stage of the decision process these actions incorporate.

Another view may be termed the "decisive progression" view. Here a progression is a sequence of actions (usually taken from a larger sequence), see [15]). Now a sequence of secondary actions is a decision (about some future activity A) provided:

- The sequence of actions causes A to take place (some theory of causality must be assumed).

[10] It seems plausible to place open source near the opposite end of this spectrum, where the extreme opposite end is occupied by control code production and release methods where besides a control code also a specification and a correctness proof which can be checked by the user is handed over from producer to user. If the check fails or if the specification is incompatible with user requirements the sale is off.

[11] In contrast the literature on risk analysis pays much attention to "what is risk", and the literature on finance has many contributions on "what is money". But in computer science one observes a similar phenomenon: questions like "what is a computer program", "what is an operating system", "what is computer program testing", "what is malicious code", and "what is a computer virus", do not attract significant attention.

- An alternative sequence of actions can be imagined which in a plausible alternative world leads to a future in which activity A will not take place (the alternative sequence of actions 'avoids A'). This alternative must be obtained by modifying actions that incorporate degrees of freedom for agents involved.

- No shorter other sequence or subsequence also causes A and at the same time allows for an alternative sequence of secondary actions that avoids A.

In complex organizations the formal decision process (thread progressions that qualify as decisions according to the formal criterion) may be so extensive that it is reasonable to refine their secondary actions recursively into secondary actions of a first and of a second class. Secondary actions of the first class represent the major formal steps in making a particular decision while the second kind (of secondary actions) may glue together to progressions of actions that at a closer inspection may be considered decisive (for making the mentioned decision).

The connection between decisive traces and control codes lies in their sharing of the property that the causalities involved determine their classification as such. For a code to constitute a control code (in the sense of [13]) its (caused) impact on system behavior is dominant over its self-proclaimed role. This observation gives room for the suggestion that the theory of control code may be used as a point of departure for development of a similar theory of decisive progressions that may underly a theory of decision making.

3 Control code testing

It should be noticed that no intellectual approach, that is no form of systematic reasoning, can exists on which to base the task of understanding or predicting what behavior of a box B is brought about by it running C, that is by executing B(C).[12]

This observation, in combination with the need to make decisions about control code release, deployment and usage gives rise to the prominence of testing. This section is aimed at a clarification of some conditions under which control code testing constitutes a plausible part of control code related decision making activities. The control code is not assumed to be safety critical in this section. A definition of safety critical control code and an assessment of control code testing in connection with safety critical conditions of usage is discussed in Section 9. In [27] a listing of four 'salient variables' concerning decision making is given:

1. possibility of continuity,
2. reversibility,
3. range of effects,
4. time pressure.

These variables are helpful to draw a general picture of the context of decision making envisaged in this section. It seems reasonable to connect safety critical conditions with reversibility. Thus reversibility of decision is assumed to some degree. The possibility of continuity is connected with the disadvantage implied by not making use of forthcoming control code. It is assumed that this disadvantage is rather high which leads to some but perhaps not acute time pressure. Finally it is important to get an adequate picture of the full range of effects of the spectrum of decisions that at any moment can be taken about some control code.

3.1 User side control code testing

The following line of reasoning at the side of a user U may be considered acceptable:

[12] The control code need not to have been programmed, let alone that it is open source. For the difference between programmed and non-programmed control code see [13] and [25].

3.1.1 Acceptance test rule (ATR).

Let B be a box of type t = t(B). Let C be a CC that is an executable for boxes of type t. ATR allows to conclude from the following 11 conditions to infer the subsequent 3 conclusions.

1. C has been manufactured by a trustworthy control code manufacturer M (in particular trusted by U), well experienced with boxes of type t(B), and if,
2. the website of M writes that it produces implementations of a service S, which is also written on the limited documentation of C, and if,
3. test specialist T is aware of service S and the utilities a sound implementation may bring with it (for U), as well as the disutilities that an invalid implementation of S may create (for U), and if,
4. running B(C) on a sample input, as T has been doing, according to T indeed provides the service S (that would bring, according to condition 8 below, significant utility UT for U when performed by way of usage instead of test), and that compliance is persistent along a number of runs (tests) with different inputs (which may be provided interactively rather at once), and if,
5. T agrees with the views that M has about Service S and the task of its implementation, and if,
6. S has been automated on control coded devices of type t(B) before, and if,
7. user U of box B trusts test specialist T and if,
8. U needs service S, and if,
9. nobody trusted by U and easy to consult about the matter is knowledgeable about how B(C) technically works, and if,
10. the utility UT of consuming service S to U is substantial, whereas
11. the disutility of a failed service (intended to provide S) is low.

Then it is plausible for U to perform the following 3 inferences and actions:

1. to infer that B(C) delivers S on all or most relevant inputs, and,
2. to acquire the right to use C on U's box B, and,
3. to decide (conditional of the acquisition of the right of using C for the purpose of providing S) to make use of B(C) when service is is required by (U).

3.1.2 Rationale for the acceptance test rule.

ATR is an expression of reasoning not included it ATR's statement itself. We will highlight this reasoning and thereby explain the presence of each of the mentioned conditions. The primary rationale for ATR is that it would be implausibly accidental for C to deliver S when B(C) is executed. The risk exists that U (or T on behalf of U) is being deceived by M but that risk can be ignored because U trusts M. M publicly claims expertise in producing control code for service S on boxes of type t(B). This confirms ex post plausibility that C is a product of M. T is able to test for delivery of service S and knows what is at stake when its judgements are wrong. T knows abut 'implementations' of S on t(B) type boxes and in particular (by keeping track of professional literature) T knows about M's strategy for realizing S and T agrees with that view. Thus T is not struggling against its disbelief in the general approach taken by M concerning service S. U has nobody else to rely on who is more knowledgeable about the technical details of what is going on when B(C) is running. Because U needs S it will rely on T's advices in the matter. U's potential gain (utility of using C) is substantial, while his loss (disutility when, in spite of the preceding facts C, fails to deliver S on box B). Here ATR may be overly restrictive. In some cases U would go ahead even if the expected disutility of failure of C to deliver S on B is significant.

3.1.3 Ex ante control code testing for users.

ATR provides an argument that is only useful for a prospective control code user who is in need of an ex ante confirmation that the plan to make use of C is reasonable. Because in general one assumes a wide gap of expertise between user and manufacturer it is plausible that a user works and thinks according to the rule, provided he assumes the ability of a manufacturer to produce sufficiently errorless control code.

The prospective user will base its decisions on functional tests, which are equally informative about processing times. The tests performed on behalf of a user might be termed acceptance tests because U may decide not to acquire usage rights if T is unconvinced. Strictly speaking ATR belongs to single box control code logic because no other boxes are mentioned. It is, however, implicit that U assumes that M has successfully performed tests on C with one or more similar boxes B_1, B_2,... of his own. The plausibility of the acceptance test rule depends on the existence of multiple boxes and assumptions about executions of comparable control code made on other boxes.

3.2 Producer side control code testing.

The intuitive obviousness of testing for a user is suggestive of the option that testing also works for the manufacturer in ex post mode of operation (after the code has been designed and produced). But this argument is flawed. The arguments leading a control code producer to perform tests must be quite different in structure.

A well-known, but by no means generally accepted, argument forcefully promoted in [3] and pioneered in Beizer's [6] test process maturity model, suggests that control code testing ex post at the manufacturer's side is not functional but must be structural. The purpose of structural testing being exclusively to improve or validate the control code production process, and not to improve product quality in any immediate fashion.

According to [3] functional correctness must have been assured by adequate engineering methods, involving requirements engineering, stepwise refinement, software design, program translation (compilation), and control code generation. Structural testing takes the functionality as given and focusses on the rationality of its implementation in an instruction sequence. Structural testing is performed with the knowledge of the structure of the code before its compilation into executable form (relative to type t(B) boxes). Thus assuming that C is a program (it has come about[13] via translating an instruction sequence), that instruction sequence can be analyzed (verified, simulated, model checked, submitted to code walkthrough, peer reviewed etc.) in order to detect and remove design errors (including the so-called programming errors) as well as errors of requirements capture. Structural testing belongs to this family of activities.[14]

3.2.1 Release test rule (RTR).

CC manufacturer M may use the following rule before deciding to ship CC to the intended user community. Shipping precedes the acquisition of licenses by users because users will be allowed to apply ATR before deciding to make use of the CC. The rule is called RTR because it is supposed

[13] Or rather: the most plausible conjecture about C's genesis it that it has been compiled from a preexisting instruction sequence

[14] It may be even reasonable to claim that: in the absence of information about C's production process no ex post testing at manufacturing side produces any substantial degree of confidence.

to take place just before a (new) release of the control code is issued, while many intermediate tests may have been applied during production.

RTR: Suppose the following 7 conditions are fulfilled,

1. a box type t is given,
2. C has been manufactured by M by means of a production process well-known to M on the basis of requirements (R) that were available in advance, and if,
3. these requirements provide a clear set of hypotheses (hereafter called SPEC for specification) which specify what is expected from executing C (after successful production), and if,
4. C is a compiled polinseq (say P) such that either
    - P has been systematically reviewed (against SPEC) according to procedures clearly documented in M's CC production process description, or
    - P has been verified (including model checked) by means of documented techniques well-known to M such that M is competent it the use of the tools needed for these techniques,

    and if,
5. M either
    - has performed a substantial number of (successful) tests on C using boxes of type t, similar to the tests which he expects the prospective users to perform to the extent that according to the judgement of M user side test primarily deal with the problems that may arise if their executing boxes differ from the boxes M has been using for testing purposes, or
    - has performed a number of simulations if boxes of type t have not been and cannot be made available to P,

    and if,
6. there are no further persons around to whom M has easy (but perhaps compensated) access, whose advice concerning
    - C and P and the instruction sequencing notation used for writing P, or
    - the original requirements R and/or the process of its capture, or
    - software process used for writing C, or
    - the validation and testing process, that has been applied, or
    - expected disutilities at user side when C fails to deliver, or
    - expected box types at user side,
    - competing products (CC's for delivering SPEC according to R on boxes of type t) from one or more other and independent manufacturers,

    should be sought because of their intimate knowledge of one or more of these aspects, and if,
7. the business case for shipping C is clear to M, and so is the risk analysis involved.

Then it is plausible for M to perform the following 4 inferences and actions:

1. that B(C) delivers S on all or most relevant inputs, for boxes of type t at user side, and
2. to ship C to interested users, and
3. to offer them terms for acquisition and use, and
4. to communicate the recommendation to potential users to make use of C.

4 Definitions for code controlled boxes

Using the terminology of [7] an IDBR (informal description by roles) of a code controlled box can precede more formal so-called imaginative definitions. An IDBR is not unique, many alternatives can be conceived.[15] An IDBR at the same time contains more informal meaning and fewer inter concept relations than a so-called conceptual model in the sense of [39].

In the previous section of testing almost no assumptions have been made about code controlled boxes, except that use and test can be distinguished. In order to present more specific arguments about code controlled boxes a more refined definition thereof is useful.

Below a proposal is worked out for an IDBR of a code controlled box. In this definition control code execution architecture is used as an alternative phrase for code controlled box. In [14] a control code execution architecture is termed a synthetic execution architecture in order to emphasize that the control code constitutes a true part from which the execution architecture has been synthesized rather than an abstraction of some part which is only used for its analysis.

4.1 Restating the question

Before turning to the definition of code controlled boxes in more detail it is useful to state once more and in more detail which questions are to be addressed by developing an informal control code logic (ICCL), and why that is supposed to be of any importance. By introducing ICCL a rather specific level of abstraction is created for discussing computers. At this stage it is unclear which aspects of computing can meaningfully be dealt with at the level of ICCL. Thus at the same time a level of abstraction is created and questions about what may be expressed at that level arise.

The following listing of issues intends to clarify in more detail the specific properties of an ICCL level of abstraction and in addition it clarifies questions that must arise once such a level of abstraction is taken seriously.

Fundamental information hiding for computing. Understanding a computer as a code controlled box achieves a level of abstraction where no assumption is made that a mathematical or a logical theory from which to predict machine behavior is available. A user making use of a code controlled box needs to trust someone else in order to gain confidence that this is a meaningful action because he is unable to acquire any substantial confidence in the available technology by looking into its structure and details, for the simple reason that this information is hidden. This form of information hiding seems to be fundamental for modern computing where the large majority of users users cannot conceivably pretend to understand how their boxes arrive at their observed behavior.

Technology adverse computer science. I am interested in finding out which parts of computer science, or rather of computing and usage of computer technology, are accessible for those who do not even start to pretend to know how control codes have been produced and how their execution works. My assumption is that by analyzing computing technology in terms of the production and usage of boxes and suitable control codes and by explicitly hiding any information about control code production technology as a substitute for means to predict system behavior a picture of computing may be found that is realistic in the sense that it is not based on the erroneous assumption that the people involved understand what they claim to understand just because of a long standing but fully unwarranted tradition of making such claims.

ICCL as an ideology. I will assume that as a methodology ICCL refers broadly to a state of knowledge and a pattern of reasoning employed by one or more individuals working in information technology who do not accept in principle and against all odds of the omnipresence

[15] In [30] an IDBR for an operating system is provided, while concluding that more informative definitions of operating systems are unavailable.

of so-called experts that they can base their activities and carry their responsibilities on any hypothesis about an intellectual understanding of how computers work. So what activities and tasks are accessible from a perspective of ICCL?

- Can one perform network and system administration (NSA) on the basis of ICCL?[16] By including a paper on instruction sequences in [9] I have committed myself to a negative answer to this question. That was premature because no hint of a proof that ICCL is insufficient for NSA has been given in [8].
- Can one (acting at the ICCL level of abstraction) be a safe private computer user sufficiently invulnerable to the problems carried around by malicious control codes? Can anyone understand at all what malicious control code is at the abstraction level of ICCL?
- Can control code production processes be managed with ICCL based competences only?
- Can control code deployment processes at the user side be managed and performed on an ICCL basis?
- Control code usage provides services to its users: can these services be outsourced by managers whose technology awareness is not beyond ILLC?

Answering these questions is far from obvious. Here is a fragmented working hypothesis that results from the assumption that positive answers on these questions may be found:

- A large and coherent fraction of IT can be based on ICCL.
- ICCL based IT at a research level can be called “behavior oriented computer science”. I suggest that behavior oriented computer science supports ICCL based IT in the same way as computer science supports IT.[17]
- ICCL based IT is a real competence in the sense that full awareness is required of what aspects of IT an individual, who is dealing with that particular form of IT, does not know and is not likely to get to know in any meaningful detail before having made the major decisions and having performed the main tasks related to that part of his task.
- Managing trust lies at the basis of ICCL based IT rather than developing understanding.

What is a computer? When moving towards a definition of code controlled boxes it appears that the concept of a computer cannot be taken for granted. At this stage I have no answer on the following questions:

- Is the concept of a computer available to ICCL or is ICCL positioned at a level of abstraction where the distinction between computers and other seemingly similar devices cannot be properly made.[18]
- Is ‘computer science’ as a phrase still connected to the concept of a computer or is it simply another (and perhaps somewhat outdated) phrase for IT?

[16] This kind of question can be compared with the question to define a health care system where only a limited subset of people involved are supposed to have any (bio) technological understanding of how molecular code based drugs work inside a human body. Which health care roles are accessible for persons who insist on not at all having such knowledge?

[17] Behavior oriented computer science must not be confused with behavioral computer science which should refer to that part of computer science which puts the main emphasis on the fact that human users apply and produce computers so that human behavior transpares though computing. Behavior oriented computer science is supposed to base its ontology on behaviors rather than on the underlying structures from which behaviors emerge.

[18] One may compare: (i) important aspects of brain science may not be accessible from the more limited perspective of (behavioral) cognitive science, (ii) technical questions about combustion engines cannot be properly posed in the vocabulary of public transportation, (iii) architects need to know what a building is, but in some cases they may be uninformed about critical aspects (and the corresponding jargon) of civil engineering.

- If computer science as a description of a scientific activity is based on the concept of computer then what distinguishes that class of devices form the many other products of electrical and electronic engineering. If such a distinction cannot be reliably made: is the phrase computer science to be replaced by another phrase such as information technology.
- Assuming that a convincing definition of computers can be provided which underlies computer science (if that is considered a necessity): is some form of universality of the mechanisms involved the critical factor for this definition or is mere flexibility in the dependence of machine behavior from control code sufficient. At present I suggest the following viewpoint as being plausible:
  1. Computer science is mainly concerned with code controlled boxes that are expressively universal according some qualitative concepts of computation. In addition:
     (a) Computer science needs a clear story about universality concerning computation. The currently prevailing story is the theory of Turing machines.
     (b) For Turing machines, however, the distinction between control code and data code is quite arbitrary.
     (c) The viewpoint that a Turing machine is executing a control code that occupies part of its tape is unconvincing.[19]
  2. A notion of universality for boxes is only meaningful for code controlled boxes. A notion of universality can be developed in terms of ICCL without any dependence on additional computational mechanisms or principles.
  3. Without some claim to universality of its boxes no significant demarcation of computer science can be claimed within mechanical engineering (mechatronics), electrical engineering, optical engineering, quantum computing technology and mixtures of these.
  4. When forgetting about universality:
     (a) even if code control is a feature of decreasing importance after all, it won't be easily dropped in a computer science context, because
     (b) non-code controlled boxes are primarily studied outside computer science,
     (c) the phrase 'computer science' is remarkably resilient against the critique that means and ends have been confused,[20]
  5. Universality implies that a code controlled box may contain a control code production environment suitable for generating control codes for itself.
  6. Universality therefore implies that its memory can store polyadic[21] control codes which result from its own activity and that moreover execution of a stored polyadic control code is itself achieved as the effect of an available primitive action which in its turn is a possible effect or side-effect of control code execution.[22]
  7. At this stage matters become confused. Using Kleene's S-n-m theorem from recursion theory as soon as a (code controlled) generalized box can compute all computable functions it also computes a universal computable function which might be

[19] The concept of a Turing machine as a device made for executing instruction sequences (programs) is also unconvincing. It is unclear why some tape contents ought to be considered instructions and other tape contents data.

[20] As if combustion engine technology could stand for automotive technology.

[21] "Poly" here refers to the plurality of items available for successive usage. So polythreading is about several threads operating in succession, while multi-threading refers to several threads operating concurrently. Multiprogramming as opposed to "polyprogramming" is about producing a plurality of programs that determine threads which are run in a multithreading regime. 'Polyadic instruction sequence' is used in [11] because 'instruction sequence' fails to have the flexibility of referring to a plurality of instruction sequences. Polyadic control code refers to plurality of control codes that may together fulfill the role of controlling a box.

[22] This latter requirement goes beyond the specification of an AnARCH (analytic execution architecture) in [14].

considered an interpreter for stored polyadic control codes. But there is a difference between interpretation and execution and the capability of a box to execute rather than to interpret a stored polyadic control code cannot be inferred from the S-n-m theorem.

Of course claiming plausibility for the above points of view is only marginally different from formulating all of this as additional questions.

A continuum of definitions. In [7] an exposition of methods of definition has been proposed. The optimal form of a definition is supposed to be a so-called imaginative definition in the form of an LSCD (logical solitary concept definition). Initial stages of definitions are found as IDBR's (informal descriptions by role). For a concept given by an IDBR usually many different and competing IDBR's can be conceived and many paths towards more informative LSCD's can be developed. Some remarks on this matter:

- When defining the concept of a code controlled box several difficulties emerge. The definitions given below do not provide completely satisfactory remedies against these complications. Instead these definitions may be criticized in several ways and thorough revisions in subsequent work cannot be excluded because of the inherent difficulties of the issues.
- A code controlled box is a device which is characterized by a significant dependance of its behavior from its (polyadic) control code. To define a box in all detail requires providing information about this dependence. But providing complete information is rather unfeasible.[23]
- If someone working and thinking at the level of ICCL needs predictions about control coded box behavior an additional specification for such predictions needs to be provided. Technical information about the control code production process is no substitute for this specification.[24]

4.2 An IDBR for code controlled boxes

A code controlled box B is characterized by the following features and properties:

1. B has a user interface with elementary controls like switching on and off, and for activating the features mentioned below. These controls are called operator controls and they are handled by the operator (operator is considered to be a role of the user if one prefers to think in terms of users).
2. B features a mechanism of code exchange (for a classification of such mechanisms see 1.3).
3. B can be loaded with one or more codes (conceptually known as bit sequences).
4. B has a limited number of modes of operation that can be activated from its user interface where the major modes can be understood as: producing the behavior expected from executing the loaded control codes and data by box B. At this stage there is no preferred theory of behavior. In the synthetic execution architectures of [14] (a possible form of box) only a

[23] The issue may be compared with focusing on a specific human being while ignoring the molecular biology of his brain. One may 'define' a human agent role as a role to be performed by a human being while not taking into account on purpose any link between the expected behavior of this person and the physiological state of his brain. Specifications for his role as well as expectations about his behavior need to be provided at a higher level of abstraction.

[24] An attempt to clarify the distinction between a mere code controlled box and a box where behavioral prediction is enabled has been made in [14]. In that paper a synthetic execution architecture describes 'what it is', while an analytic execution architecture describes how a machine may work, or rather how it might conceivably work, with so much detail that behavioral predictions can be derived.

single code is loaded and its (only mode of) execution gives rise to a behavior specified as a process in process algebra. In the machine function based model of [13] (another conceivable form of box) a sequence of codes can be loaded and a mathematical function returning a sequence of codes describes the behavior.

5. If several codes are loaded a distinction between control code and data code can be made or be sought after. For machine functions that has been worked out in detail in [13].

6. If more than one code is classified as control code the control code is called polyadic. This may depend on the mode of operation chosen by operator control.

7. B may be viewed as a tool for executing control codes already known from their roles in the context of other boxes. Alternatively control codes can be viewed as tools for making B work. Both views may co-exist for the same box and codes.

8. Viewing a control code as a bit sequence, a plurality of control codes can be simultaneously in charge of the behavior of a box as well. This is called a polyadic control code.[25]

9. By executing a polyadic control code a box produces a service. Making use of that service is the most obvious reason for an operator to initiate that execution.

10. Polyadic control codes may but need not have been produced by compiling polyadic instruction sequences that serve as an intermediate design stage. At the level of abstraction of a code controlled box nothing is known about the production process of (polyadic) control codes and no theory is given that predicts the behavior during execution from the structure, content or form of the (polyadic) control code.[26]

11. B is equipped with one or more control code exchange mechanisms as have been specified above. These exchange mechanisms can be directly controlled by an operator. A specification how that works is contained as a part of box B.

4.3 Analytic execution architectures for code controlled boxes

The IDBR for a code controlled box of the preceding section is unclear about how the transformation from polyadic control codes to box behavior exactly works. If nothing is known merely a type is described of which boxes are instances. If it is felt necessary that a code controlled box provides more information then in addition an abstract analytic code controlled box can be given as follows:

1. A parsing and syntax analysis operator D (disassembler) is introduced which transforms polyadic control codes C_p for B into structured data, called polyadic structured control code.

2. Volatile data structures specify data that are maintained by a state during the analytic execution of a polyadic structured control code. After each termination of an execution the volatile data are reset to the same initial values. Permanent data structures maintain information that remains unchanged by termination of an analytic execution.

[25] A number of instruction sequences allowing jumps between different sequences has been termed a polyadic instruction sequence in [11]. The situation with control codes is comparable though different. First of all, in contrast with the notion of an instruction sequence, according to some the notion of a control code may have the flexibility of consisting of more than one bit sequence. Secondly the internal relation between different parts of a polyadic control code is unclear. Thus unlike the notion of a polyadic instruction sequence the idea of polyadic control code is in need of additional explanation.

[26] If it is known how to disassemble a polyadic control code into a polyadic instruction sequence then using the terminology of [14] an analytic execution architecture can be used to explain the behavior generated by the formal execution of the polyadic instruction sequence which at the same time models (predicts) the behavior created by box B when executing the mentioned polyadic control code.

3. (Analytic) execution of a polyadic structured control code is mathematically defined by means of operational transition rules that make use of structured operational semantics concerning the structured control codes. The adjective analytic for an execution is use because it is at a higher level of abstraction than the reality it is supposed to model.

4. It is required that disassembly followed by analytic execution in the context of initialized volatile and permanent data structures produces the same behavior as an execution of the original polyadic control codes on the given code controlled box. This requirement can only be understood is some mathematical definition of the effect of polyadic control code execution the given box is known in principle. The intuition for ICCL is that although such a definition exists in the mathematical universe it is considered to be of no practical use because it is too complex. In software engineering terms the definition can be considered a hidden part of the specification of the box.

5. It may be maintained that any code controlled box B must be equipped with an abstract[27] analytic execution architecture AE_B as specified in the above fashion. At least the producer of C_p has it at hand.

6. In general AE_B is supposed to be hidden from the users of C_p and for all end users served by those users as well. Thus users know that executing C_p on box B produces a behavior which is given (can be predicted) by an analytic execution of $\mathrm{D}(\mathrm{C}_p)$ on AE_B but they don't have any practical means of accessing these predictions. Their only way to obtain such information is by actually executing C_p on B.

4.4 Intrinsic universality for code controlled boxes

If a code controlled box is given by a machine function according to the above IDBR, then it is reasonable to view this alternatively as a so-called synthetic execution architecture (SynARCH) in the sense of [14]. The adjective synthetic means that it synthesizes parts that model components of an actual or imagined system rather than abstractions thereof. Opposed to a synthetic architecture [14] proposes the notion of an analytic execution architecture (AnARCH). Instead of an AnARCH a generalized version of it AnARCH_g has just been outlined.[28] The components of an AnARCH are abstractions of what one expects to find as components of a box in practice. The key advantage, however, of an AnARCH is that it explains (and for that reason predicts) the results of execution which a SynARCH merely lists in the graph of a mathematical function.

In order to make AnARCH_g produce a model of the computation of C_p on B the control code C_p must be disassembled into structured control code. Now it may be the case that the disassembler projection produces structured control code that has certain parameters constrained by numerical bounds due to the finiteness of the collection of executables for the given AnARCH_g. For instance the number of codes, their length and the number of occurrences of different features used, may be limited to certain bounds. These bounds may be linked with bounds concerning the data structures that span the state space of AnARCH_g.

In principle one may imagine that the bounds for an AnARCH are considered to be artificial. Then removing these bounds may be considered a natural generalization step. Such removals may be performed in different ways. It is supposed that a natural generalization of AnARCH_g say AnARCH_G is obtained. If there exists an encoding of natural numbers in inputs and outputs that can be processed and produced by AnARCH_G and if all computable functions on natural numbers can be expressed by means of polyadic structured control code for AnARCH_G, then AnARCH_G is said to be computationally universal. That qualification is extended to AnARCH_g.

[27] The reason to speak of an abstract analytic execution architecture is because there is no implication that disassembly of C_p will produce a polyadic instruction sequence, a fact that is assumed in [14].

[28] The AnARCH is a special case of an AnARCH_g where the polyadic structured control code is required to be a polyadic instruction sequence and the volatile and permanent data structures are represented by means of services (see [12]).

Now code controlled box B (a SynARCH according to [14]) is said to be intrinsically computationally universal if there is a very plausible AnARCH$_g$ together with some disassembler transformation D for the executables of SynARCH explaining the behavior of SynARCH relative to some AnARCH$_g$ such that a very plausible general version AnARCH$_G$ found by removing (artificial) numerical bounds on the features used in D(C$_p$) has universal computational power.

4.4.1 Definition of a computer.

A computer can be defined as a code controlled box which has intrinsically universal computing power in the sense just explained.

This definition is ambiguous in the sense that it depends very much on what is considered natural. Further if structured control codes have been made up for the purpose of motivating that a given box B is classified as being of universal computational power no subsequent notion of naturalness for removing its numerical bounds is easily put forward in a convincing manner. But the definition can be applied if it is assumed that it is used only to demonstrate that a given box (intrinsically) has universal computing power (when considering a specific proposal for its generalization by removal of one or more finite bounds) while it is not used in cases that a positive judgement that matter is refused (that is, no unsuccessful quantification over natural generalizations is assumed, only the lack of such a proposal is reported).

5 Control code risk analysis

Both the user and the producer of control codes can be confronted with risks. Risk analysis may but need not be invoked as a part of decision making. I will try to draw some picture of decision making and how it may occur in connection with decisions about release, deployment and usage or control codes.

Needless to say risk analysis is a large subject by itself but an attempt to highlight aspects of primary importance for the production and use of control codes is legitimate.

It seems to be the case that user side risk analysis is only possible on the basis of a strategy of use which must be known beforehand. If a control code cannot be upgraded after an operational failure thereby ‘solving the problem’ the context clearly differs from a context where that can be done. If a user coordinates the activities of many end users the issue of risk communication appears which will otherwise trivialize.

However, providing a full survey of operational models for the use of control codes is unfeasible. After some general remarks about risk analysis I will consider the user perspective from the assumption that the user will put the control code at work to serve a community of end users. Thereafter the producer side perspective will be discussed from the perspective of a producer which provides control codes for the same type of system. The producer is supposed to have a production workshop in place which keeps producing improved and modified polyadic control codes for a rather narrowly defined and coherent class of systems[29]

5.1 Control code related strategic risks

Both the user and the producer may have control code related assets. For the user this is the control code itself plus the result of investments made in preparing its use. For the producer the assets include the production capacity for these control codes as well as knowledge about it, knowledge about competing products, knowledge of actual and potential clients, and reputation amongst them.

In [20] strategic risks are summarized based on Simons [37]. It is concluded in [20] that Simons' division in three forms of strategic risk applies to real estate. I put forward that with some minor

[29] Here ‘system’ refers to expected behavior of a box executing the PCC.

adaptations this analysis of strategic risk applies to control codes usage and production just as well. Simons defines strategic risk as:

> an unexpected event or set of conditions that significantly reduces the ability of managers to implement their intended business strategy,

and he distinguishes the following risks (where I have drastically shortened and simplified the descriptions):

Operations risk. The user may fail to organize control code deployment deployment and efficient use. The producer may fail to meet his objectives due to management mistakes and organizational flaws.

Asset impairment risk. The control code may fail to deliver the required service because of changing circumstances. It may fail to run on a new generation of boxes. The production process may become outdated with new regulations or quality control mechanisms becoming prominent. Or key personnel for production may be leaving.

Competitive risks. Competitors may render the use of a control code or the production of modifications and upgrades of it futile.

Competitive risks are not very special for the case of control code except for the fact that in case control codes are not protected by patents or copyrights and these are produced for and sold in a general market, such risks at producer side can appear overnight. This is due to the minimal cost of transportation and reproduction. User side competitive risks are primarily related to the user's core business which is more likely than not outside control code production.

Asset impairment on the other hand is a non-trivial phenomenon which needs to be taken into account. It is important to notice that qualifying a control code problem as an instance of asset impairment may be controversial. Many external observers have classified Y2K problems as asset impairment problems to be remedied with reparative actions. But many control codes had on purpose been designed so that their functionality was only guaranteed until Y2K. The counterintuitive property of control codes is that unavoidable depreciation may take place without any change occurring in the control code itself. Because such depreciation, if coming from known time limits, is extremely predictable it is not justified to qualify it as a risk as risk must be connected to uncertainty in the first place.[30]

In the sequel of this section operations risks will be discussed in more detail. Because uncertainties are most obvious at the user side attention is limited to that side. Risk, or a plurality of risks may exist without U being aware of that fact.[31] It is this observation which Rosa mentions as a decisive argument for a realist perspective on the concept of risk.

[30] It is even reasonable to assume that before a systematic and specific analysis has been performed the uncertainty of risks cannot be analyzed in probabilistic terms based on generic and known models either. The more random an outcome of some process is, the less reasonable is to speak of it as a risk, unless the probability of that outcome is very small in which case it might constitute an acceptable risk because it lies below some predetermined threshold. The use of terminology of risks is rather intricate and as it stands it seems to be the case that some risks are not immune against risk analysis. Once mechanisms become known and probabilities are clarified some risks become usual system properties and for that reason cease to be analyzed in terms of risks. A terminological difficulty is that who speaks of 'taking some risk' does not by that speech act alone either introduce a risk or discover an already existing risk or even prove the existence of a risk (according to risk theory). Who 'takes a risk' usually has already performed risk analysis and is now taking a chance in a context with known and different rewards. Problems that have been frequently observed cannot be considered materializations of underlying risks. The primary intuition of a materialized risk is a course of events that has not yet been observed and which causes adverse consequences. So one may speak of the risk of the USA being unable to settle its governmental debts within the next 25 years because it has not happened before and probabilities are hard to assign. Risk analysis regarding this matter may clarify consequences with relative probabilities for (business plans of) other countries or enterprises or even for individual citizens inside or outside the USA, but it will not do away with its status as a risk.

[31] Rosa [35] suggests that the risk which exists independently of any observers may give rise to three epistemological stages for agents or groups of agents involved: awareness, analysis and assessment (quantification or qualification of probabilities and impact), and management (defining and executing a policy).

5.1.1 Risk materialization and risk identification

A risk is said to materialize if the sequence of events that constitute its 'body' take place. Materialization of a risk need not imply that the adverse consequences (damage) which justified the qualification of an uncertain conceivable course of (future) events as a risk are actually inflicted on those who were at stake. If so the risk has materialized with the feared damage (or with more/less than the feared damage), if not the risk has materialized without damage. The latter may have been a consequence of successful risk management.

Once a specific risk has been identified a measure of severity of the risk yet has to be attributed. Obtaining such information is a major objective of risk assessment. Risk assessment is an activity aimed towards obtaining additional information to the level that probabilities or probability intervals replace mere uncertainties, and to the point that various outcomes have been valued in terms of their disutility (damage). Risk assessment may be quite straightforward (at least conceptually) in which case it follows a detailed and known sequence of steps. Following [17] not all uncertainties can be understood as probabilities although many authors insist that uncertainty and probability unavoidably correlate. Risk assessment may also be much more scientific in nature, involving new and dedicated research projects, or it may be social. For instance public outrage created from an adverse outcome can be classified as a risk because it is real. It can sometimes be predicted from an investigation about risk perception in advance.

Risks may be so hard to assess that scientific research is not likely to shed light on them soon. In some cases the risk might be high because of the sever adverse consequences that can be imagined, while assignment of probabilities or any other form of quantification of the risk is beyond current scientific capabilities. For such circumstances precautionary policies have been developed. But, it is worth noticing that as a decision rule precaution is incoherent according to [33].

5.1.2 Risk analysis paradox

In order for an agent A to determine whether or not a significant investment in risk assessment and in subsequent risk management is needed (or justified) before making some decision D it is necessary that A knows that such risks (caused by effectuating D) may exist in principle. Here is a bootstrapping phenomenon: some faint awareness of a risk is needed beforehand to justify a subsequent non-trivial risk assessment.

This 'paradox' has its counterpart in risk research. Although in theory the perfect application of risk analysis will detect novel risks before they can lead to danger[32] and risk research should deliver the tool for risk detection, a survey of risk research literature gives rise to the presumption that for risk research just as well as for risk analysis practice the awareness of a risk needs to be triggered by adverse human experience before systematic research efforts are made in a particular direction.

5.1.3 Risk management paradox

A focus on risk analysis and assessment may hide, and even prevent a thorough, professional, and state of the art approach to quality control regarding a control code. It is not justified to view the

[32] Following Rosa [34, 35] a danger is a risk with a high probability of leading to damage. In the light of previous remarks this definition is manifestly unsatisfactory because precisely the high probability constitutes an important argument against the use of the term risk. Perhaps the terminology can be made consistent by adding the category of pre-risks which combines risk and dangers. This issue is about knowledge. If the pre-risk is known to materialize with a high probability it is a danger and not a risk (and known not to be a risk), if the same probability is high and if that fact is not known the pre-risk is a danger and it is also a risk, but is classification as a danger cannot be performed. However, if the pre-risk has a high probability of materialization and that is known it may still be considered a risk if the materialization has never before been observed. Thus: (i) known dangers are not risks unless they constitute novel phenomena, and (ii) identification of risks which are dangers as well is the primary task of risk analysis.

consequences of quality problems as risks if it is known how to detect and how to prevent these quality problems.

If risk analysis is preached in the context of control code both control code errors caused by instruction sequencing errors (assuming that the control code has been produced by compiling an instruction sequence) and control code errors caused by design errors for the underlying instruction sequence should be considered with hesitation as potential causes of potential risks. These problems may have been introduced by a negligent instruction sequence production process with defective quality control. Subsequently paying lip service to risk management is no adequate compensation for that kind of negligence.

Consequently: (in)validity risks (that is risks emerging from adverse consequences of control code production errors) should only be investigated after it has been established that control code production has been performed at an adequate quality level. An estimated high probability of control code invalidity may constitute an incentive to ask for higher production quality standards from the control code producer. Indeed risk analysis for control code usage in a specific case requires the determination of the quality level for control code production. Making sure that this quality level has been achieved is primarily a task for the producer. If the suspicion exists that a control code defect may directly or indirectly cause bodily harm of any kind as a consequence of executing said control code, that control code is considered to be safety critical.[33] Users of safety critical control code are entitled at any time to be informed about the control code production process for their code. Users need to be able to convince themselves that this production process is up to date in principle and that it has been performed adequately in a specific case. In the context of safety critical control code user side testing cannot be a substitute for a quality assessment of the control code production process, either by means of process inspection or by means of reputation analysis.

5.2 User side control code risk analysis

Control code usage can lead to significant and unexpected disutilities which are in some cases best viewed as materializations of risk. Risk can be defined in many ways. I will now follow Rosa [34], subsequently elaborated in [35], who defines risk as:

> Risk is a situation or event where something of human value (including humans themselves) has been put at stake and where the outcome is uncertain.

The objective of this definition is to make risk a property of the real world, independent of subjective judgements. Thus following Rosa I will assume risk to have metaphysical content rather than epistemological content. Taking a less principled perspective Althaus [2] provides a survey of diciplinary contexts from which the concept of risk may be understood. When U decides to make use of control code C for some specified period of time some uncertainties arise:

1. Will there be any forthcoming execution of control code C on box B initiated by U which has unsatisfactory consequences and where the problem should be diagnosed as a defect of C (rather than a defect of B or rather than as a mistake made by the operator in deciding to execute C on B at that moment and for that purpose).

2. If there will be a forthcoming execution of C where due to a defect of C ‘something of human value’ was at stake and was damaged: is a listing of these matters of human value known in advance (that is when U decides to acquire the right to use C and decides to start using C) or is there an aspect to this damage that has in no way been foreseen.

3. If an unforeseen consequence of executing C (considered non-defective until that moment) takes place and this consequence is considered ‘severe’, will it be the case that retrospectively (ex post) U would have been well-advised to put more efforts in risk analysis (ex ante).

[33] We will deal with safety critical control code in more detail in Section 9 below.

4. If the moment arises that U concludes that a more intensive and effective risk analysis for the usage of C for its own objectives had been necessary: is it reasonable to assume that a stronger effort to detect the risk that eventually materialized in the form of damage in advance would have been successful.

In the absence of conclusive information about these matters U must in any case ask the question which damage might be caused by various conceivable defects in B and/or in C when execution takes place.[34] If U sees no potential damage, or only risks with low severity he may decide not to embark into a subsequent and cumbersome risk analysis.

5.2.1 Risk levels for control code deployment.

The following risk levels for the deployment of a control code C can be distinguished:

- safety critical (risk to health and life of human beings),
- business critical (risk of enterprise going bankrupt),
- annual profit critical (risk of writing red figures),
- employee satisfaction critical (risk of loss of employee satisfaction),
- customer satisfaction critical (risk of loss of customer satisfaction),
- customer loyalty critical (risk of loss of customer loyalty),
- estimated amount of money critical (risk of loss of amount X)

To each of these risk levels a corresponding and equally named quality level for control code production is coupled and a minimum required product quality level can be determined. Therefore user side control code usage risk assessment can be replaced by the following consecutive steps:

1. spotting one of these levels for each potential usage (of C on B) that may take place under the responsibility of U, and determining an upper bound H of these levels for the full range of conceivable usages of C (by U).
2. checking that C has been produced with corresponding quality level corresponding to H or with a higher quality level.
3. performing a theoretical analysis that service S as specified in advance (on the basis of requirements capture) to be what C provides when executed on B will not be performed in such a way that damage corresponding to its risk level can occur.
4. if all of these steps succeed: infer that risk analysis for C by U has been successful (that is: has not revealed any problems). This suffices to conclude that condition 11 of ATR of section 3.1 is met to satisfaction, which in turn brings nearer to completion a successful application of ATR.

[34] Here we assume that a defect in B or in C or in their alignment can (but need not) cause a risk R (or a plurality of risks). R in its turn may or may not materialize (thus leading to damage and harm), because several other factors not connected to B and C with unknown probabilities may be involved in the process of the risk R becoming a threat, a danger, and finally, a calamity.

5.2.2 Heuristics for control code usage risk analysis.

For the specific case of U performing a risk analysis concerning his use of control code C on box B for a range of objectives the following assumptions may be taken into account:

- There is no need to apply the precautionary principle to its full extent. Scientific research will be able to provide an adequate cause effect relationship concerning the consequences of defects in C. However, that may be overly expensive in which case the precautionary principle may still be used as a source of inspiration for policy design. The specific form of argumentation (and in particular the assumptions on which it is based) that makes use of the precautionary principle should be compared with Peterson's impossibility proofs in [33] in order to guarantee that the argument that one actually makes use of is not flawed.

- Because the risks if any are dependent on the specific context of use, only U can provide the basis of a risk analysis for his use of C. It is impossible for the manufacturer M of C to perform such a risk analysis once and for all for every potential user of C. For that reason M cannot ever assume responsibility for the consequences of risks that have unfortunately materialized in connection with all users of his product C (provided there are many such users and M is unaware of their specific circumstances).

- Only if C has been manufactured, tailor made, by M for U, with a clear conception of all relevant use cases in advance, M can be asked to perform a risk analysis to such an extent that U is relieved from that task.

- On the basis of a sufficiently large collection of conceivable use cases U can decide to what extent a systematic risk analysis is required before deciding to acquire C and to start using it. Perhaps C needs to insure himself against liabilities in connection with this usage. Obtaining an insurance policy at a reasonable price will require that U has performed a substantial risk analysis.

- Risk analysis is independent from the question to what extent C is correct. Correctness means that its execution on B will produce a thread or more generally a process which complies with a specification that U has provided or has agreed with. Non-compliance with the specifications need not constitute a risk, and compliance with these specifications need not imply the absence of risks.

- If U executes C on B to provide services to clients from a client base CB, then agents from this client base must be asked for assistance in the first bootstrapping phase of risk assessment: each client may in principle provide use cases each featuring a sequence of events leading to adverse consequences, that is a scenario which is suggestive of a risk.

- The formidable size of the literature on risk assessment renders it very unlikely that a software engineer E can perform a professional risk analysis without systematic preparation in the field of risk analysis. However, having performed the risk analysis in a non-professional fashion constitutes a risk in itself for U. A significant time investment for reading about risk assessment in general is required for a non risk specialist E to prepare for this task.[35]

[35] This investment is needed if the risk must be managed that (i) E's performance as a risk analyst is claimed to be problematic because of an unacceptable lack of knowledge of the field of risk assessment, and (ii) in case of damage these questions about the quality of the risk analysis process as conducted by E cause additional liabilities.

There are many books and hundreds of papers on risks and risk analysis but this size of the state of the art is no indication that in the case of control code risks the analysis must preferably be put in the hands of specialized risk analysts who may have had their practical training in quite remote fields such as nuclear waste disposal or infectious disease control. Two months of full time work at an academic level of competence should suffice to get a sufficiently clear picture about the variation of philosophies on the subject of risk theory and on how a number of well-known and state of the art risk assessment activities have been performed. Having made such preparations I guess that indefensible amateurism can be avoided when performing a control code risk assessment.

- The most prominent “risk” that resides entirely within the field of control code production and usage as such is the risk that B(C) produces a service which deviates from its specifications. But this phenomenon is well-known up to the presence of reasonable statistics about it. In the absence of unexpected consequences of an execution failure (implied by “usage as such”), this problem is about finding and repairing errors in control code which should be classified under quality management instead of under risk management.
- The second most prominent “risk” concerns the deviation between specification that were made in advance of production of C and requirements that led to those specifications. Dealing with this problem is called validation and again it is not about risk but it is about ordinary quality control.

5.3 Producer side risk analysis

Providing a complete survey of scenarios with corresponding risk descriptions for the control code manufacturer is a massive amount of work (if it can be done at all) and for that reason it is outside the scope of this work. One may however make some assumptions about the business model of the control code producer which then provide a point of departure for an initial risk analysis. Here are some assumptions:

1. M accepts no liability from individual users, even if they have paid for the acquisition of a polyadic control code which M has released and sold as an implementation of certain tasks.
2. Each polyadic control code C_p shipped by M is the output of a production line which has been set up in order to produce a steady stream of improvements that take care of new circumstances and modifications which take care of different user requirements.
3. M is dependent on his reputation in the market. If that reputation is lost no further control codes can be sold (or far fewer). Market share for PCC’s for the specific family of tasks at hand is of critical importance for M. Below a certain market share the production line cannot be maintained because new customers will not trust that it will stay in business at least as long as their use of a new PCC (including its succession of enhancements and upgrades) has been planned to last. Reputation depends on several aspects:
   - User satisfaction for a vast majority of users.
   - Speedy release of upgrades when errors have been detected,
   - All users profit from the detection of errors in C_p and its variations as discovered by all other users. Users obtain regular upgrades that include repairs for problems that have been spotted by the entire user community.
   - timely release of announced improvements and enhancements.
   - Portability of C_p to a range of different platforms (that is: box types).
   - A buyer U of a PCC C_p will create a vendor lock in for some time and within that period U is dependent on the sustained existence of M or at least of the group within M which takes responsibility for C and similar output. M needs to maintain U’s confidence that the code can be trusted.
   - Sustained marketing potential: M needs a way to communicate to a large audience which PCC’s it has produced and how to acquire the rights to make use of those PCC’s. This potential is a constituent of reputation. Unless this reputation is available a new product cannot be sold to users who then know that they will depend on the fact that sufficiently many new customers will be found so that the business unit producing C_p like control codes will survive adverse times for M.

4. M is aware of other PCC's that different U's may execute on their boxes. P knows that combined or even sequential use of C$_p$ and other PCC's (for different objectives and from different manufacturers) can give rise to so-called security risks to such an extent that only the release of a fast upgrade of C$_p$ can limit the potential damage to M's reputation.

5. M is systematically experimenting with such other PCC's from al plausible sources.

6. M is aware that malicious control codes may be imported into user's boxes and that in particular if a malicious code enters a box during an execution of C$_p$ (in some cases even essentially facilitated by the use of C$_p$) in such a way that this phenomenon is repeatable, M will be held accountable for some damages that users are exposed to because of this mechanism.

All of these assumptions give rise to clues for risk analysis. M will need multiple criteria decision making in order to manage the combined risk that emerge from the risk analysis concerning each of these items just presented. In particular security risks are real in the sense that whether or not vulnerabilities exits in principle and to what extent these vulnerabilities can and will be exploited and the time scale for such events is quite hard to predict so that an assessment in terms of probabilities for such fact and events is nearly impossible.

In the light of current computer technology an assessment of security risks requires in depth knowledge of all technology involved including the mechanics of control code execution. This is a weakness of the overall organization. Much is to be gained if security can be analyzed from an ICCL perspective because in that case arguments are simpler and mistakes are less likely to be overlooked.

Concerning rule RTR all that can be said is that risk analysis for the producer can be a very complicated effort which finally leads to a yes or a no, which can be used to satisfy the relevant conditions of RTR.

6 Control code related quality management

Given the formidable importance of quality management in today's computer industry the question arises to what extent quality management can be analyzed and assessed on the basis of an ICCL perspective. This raises fundamental questions of priority and ordering:

1. Is the presence of a sound risk analysis for the decision to acquire and to use a polyadic control code (that is an application of ATR) a component of process quality, or is conversely, lacking quality a risk that needs to be noticed and subsequently assessed. (In other words: is at the user's side quality management in charge of risk management or the other way around.)

2. Is the risk that a producer fails to provide quality code a relevant concept, or is the removal of that risk implicit in the very definition of quality code.

3. Is compliance with requirements and specifications an aspect of product quality or should that be presupposed before any quality judgements are made. This is a consequential matter: by labeling something as belonging to the responsibility of quality management its importance is undeniably downgraded.

4. The alternative, and quite attractive, view is that quality measures aspects which have not been laid down in requirements and specifications.

5. It is reasonable to define the distance between two PCC's as the smallest number of edit operations (inserting or deleting a bit, inserting or deleting an empty control code) which allows to transform one of the PCC's into the other one (and symmetrically of course). An

error (sometimes called a bug) in a control code can be understood as a significant deviation of the behavior it generates from intended (specified) behavior which can be remedied by means of a modification that changes a PCC to a variation of it which is near in terms of this notion of distance.

With some effort sequential composition of such transformations can be described which requires that they don't interfere. All of this can be made explicit in the theory of string rewriting. Using these concepts it is reasonable to assign (in some cases) a number of errors to a PCC, or a number of errors that has been found and repaired (or that number in a recent period and so on).

From some stage onwards the statistics of bugs (detection and repair) is considered relevant for product quality assessment. Independent from a specific product or product line that statistics is considered relevant for general production process quality assessment. Following [3] the main virtue of maintaining bug statistics is connected with this latter purpose.

6. There seems to be an remarkable agreement in the software engineering literature that product quality for control codes cannot be properly assessed without simultaneously looking into the production process thereof. If that is true product quality cannot be assessed at an ICCL level of abstraction.

Without a clear position on what it means to produce control code according to requirements and specifications it is difficult to determine the stage from which the product or one of its versions exists and further efforts are directed towards bug elimination and quality progress.

The question whether quality management or risk management should take the lead cannot be answered in general. It seems more useful to consider a hierarchy of risks in advance with some 'well-known' risks to be dealt with by means of quality management and with an ultimate risk management category that will always take priority over quality management if it is at all applied.

7 Control code related business cases

Both ATR and RTR call for the judgement, as encoded in the conditions to the rule, that the agent taking the decision has an interest in doing so. To validate that interest a business case must be provided. Writing a survey of business cases for control code release and control code deployment in general is not the purpose of this section, it constitutes a challenge by itself, but some remarks about it are in order.

7.1 Business case development for control code acquisition and use

In the rule ATR that describes how a test can lead to a decision the condition is checked that U needs the service S provided by executing C on B. This can be validated by demonstrating a convincing business case for the use of S. The business case must cover the cost of acquisition (including tests), it must predict the required profits, and it needs to be sufficiently attractive not to be worried about residual risks that have been detected and subsequently assessed during the risk analysis phase. The business case may range from trivial if no additional costs are expected for a well-understood service to quite complex and even speculative. It is important that U is aware of the form of his business case. A complex business case needs to incorporate its own risk analysis. Here we will outline the ingredients for a business case in a complex setting:

1. U will have a variable number of end users for S as well as a reasonably stable number of support staff.

2. By putting S in place previous methods of working will become obsolete. U needs a description of the initial setting (stage zero measurement) such that the cost and problems connected with what will become obsolete can be estimated.

3. U needs to:

    - determine the composition of the required support staff (numbers, competences, organizational structure). U may need external consultants to help in this phase. At this stage U may need a temporary license for using C that permits use for fields tests (experiments about the business case) but not for actual usage. In this stage the required performance of the support organization needs to be specified (e.g. the maximum number of outstanding calls for the helpdesk, the average time to solution for an outstanding call, the means of escalation when call handling is taking too long in a specific case, the means of escalation employed when overall performance is inadequate).
    - make a plan on how to recruit the support staff, how to set up a management structure and how to provide training.
    - develop a communication plan to the end user community in U's organization and to make a plan on how to get started and make progress with using S. The plan may involve setting up user groups and expert user groups, end user involvement in support staff evaluation. Develop a plan concerning the most plausible contingencies (server failure, network failure and so on).

4. All expected costs of the preceding stages need to be clarified.

5. The advantages of using S in a stable situation need to be compared with the cost and disadvantages of pursuing old processes. At this stage both quantitative and qualitative data must be combined.

6. A judgement about the business case needs to be stated, taking the expected costs of acquisition and user side testing into account.

The above plan is applicable within a single organization. More complex cases are found when U needs to offer service S to many independent customers from the public whose interest needs to be created by means of marketing activities. Simpler cases within a single organization are obtained by deleting stages and phases from this description.

This section has been included because without a convincing business case no conclusion from test can be drawn using ATR. Writing a comprehensive statement about business cases for control code acquisition is not the objective of this section and it will require much more space. An informative text about business cases requires its inclusion in a more comprehensive explanation of the software process. A concise statement of that nature is contained for instance in Kruchten [28].

7.2 Control code autonomy

A vary remarkable fact about control code in practice is it that it has an autonomous existence of its own almost independent of executing boxes. I will use the phrase control code autonomy to refer to this independence of control codes. Probably control code autonomy can only prevail if at least some boxes allow exchangeable control codes.

In order to be sufficiently precise I will assume that boxes are developed in an evolutionary process. Boxes move through generations. Between generations differences can be significant and may depend on the logical architecture of boxes, whereas within generations differences may be high in terms of performance and in terms of other measurable properties but are moderate in terms of architectural structure.

In more specific detail: control code autonomy comprises the following aggregate phenomenon:

1. Manufacturing, distribution and ownership of PCC's[36] is to a large extent independent of manufacturing, distribution and ownership of boxes. CC's have a two dimensional classification, one dimension representing the functionality (service) a CC is supposed to provide, the other dimension being the box type the instances of which it is supposed to control. The intuition is that boxes of some type are 'universal' for a range of CC types.

2. More precisely independence offers the possibility of intra-generational inter-box transfers of CC's.

3. To a much lesser extent CC independence offers inter-generational inter-box transfers of CC's. Inter-generational CC transfers are mainly observed for CC's that are (and have been produces as) compiled polyadic instruction sequences and it requires that the CC's can be easily reengineered into polyadic instruction sequences. Inter-generational inter-box transfer represents a case of portability as formalized in [13].

4. Rather than relying on inter-generational transfer for keeping their CC's operational, CC's are adapted (by their manufacturers) to new box types as soon as these are introduced on the market. Adaptation leads to dedicated CC's. The use of dedicated CC's is restricted by license agreements and procedures.

5. Just like box types, CC types have a line of development. CC types may be a descendant of one or more ancestors. If a CC has no ancestors it is original. Descending chains for (important) CC types may extend in time far beyond the customary period of usage for an individual box, and may range over the life time of several box generations.

6. At closer inspection even with technologies that offer customers only intra-generational CC exchange the lines of development for CC types refer to polyadic instruction sequences (see [11] for that notion), from which dedicated control codes are generated by means of (backend's) of (equally dedicated) compilers (so-called cross compilers).

Control code autonomy is a remarkable concept. Here are some further observations about it:

- Ordinary printed books or newspapers typically don't feature control code autonomy, where control code might be an electronic form of the data made accessible by the book. The development of eBooks, however, which made a significant start on the consumer market about 2008/9, will lead to control code independence for books, admitting for intra-generational transfers, by way of a strict separation between rendering technology and the textual data to be rendered for reading by a human user.

- Standardization of CC's for eBooks (e.g. via XML dialects) is necessary to enable vertical CC transfers as well.

- CC independence made its start as an omnipresent feature of mainstream technology with mainframe business computing and is now also a dominant feature for personal computing. But in many embedded devices (photocamera's, clocks and watches, mobile phones, GPS equipment, cars, (microwave) ovens and other kitchen equipment, radio's, TV's, other AV equipment) CC independence is not a dominant feature (at least at the time of writing this paper).

- Every code controlled piece of equipment (box) that needs to offer only a fixed number of services, known in advance can (but need not) be replaced by a (potentially cheaper) piece of technology which is not code controlled.

[36] I will often write CC where PCC would perhaps be more appropriate. As written earlier the notion of a CC may well have the flexibility of consisting of a plurality of codes (bit sequences) in which case being explicit about PCC's is overdone and the default (of CC) could be PCC rather than CC with solitary CC as an indication for a CC consisting of a single bit sequence only.

- As it stands code controlled systems design is conceptually prior to the design of non-code controlled systems, at least beyond a threshold level of complexity of the system. Non-code controlled systems are simplified products derived from code controlled prototypes that constitute a crucial stage of development even when code control is intentionally rendered redundant in a final design stage. Lines of descendance for IT design are primarily visible at the level of prototypes, whereas the line of development for equipment design can be followed by visual inspection of final products.[37]

- It is tempting to compare machine types with biological species and CC autonomy with one of their features, for instance the ability to swim. In a succession of generations the ability to swim can appear and disappear and reappear depending on the ecological context. I probably cannot be asserted that by itself the ability to swim represents definitive progress, but it can perhaps be maintained that for overall evolution to be imagined retrospectively it is essential that in some stage the instances of some species (which are ancestors of today's non-swimming species) were able to swim.

- Carrying the previous remark further the development of computing machines may be considered an instance of evolution with CC autonomy representing both a potential evolutionary advantage and a potential evolutionary disadvantage for the survival and further development some machine type. This perspective renders the contemplation of a single machine representing all of computing pointless. The production and perhaps also the use of that machine will call for other probably more sophisticated and high powered machines, and the hypothesis is plausible that always some machine types, which may or may not occur with low frequencies, must feature CC independence.

Control code autonomy is very visible in computing practice, from around 1960 till today (2010) and there is no sign that its role is diminshing. The observed occurrence of CC independence in current technology raises a number of questions.

1. Assuming that CC autonomy is a feature provided by a box seller to its customers at non-zero cost: why does it survive at all?

2. What explains the important role of CC autonomy for a range of technologies (for instance the laptop), and why is it so much less important for a number of other seemingly very similar technologies (such as the digital camera)?

3. To what extent is the business case for CC autonomy connected to a market where many boxes of the same type will be sold and where many instances of the same CC type are simultaneously in circulation?

4. To what extent is the business case for CC autonomy connected with rapid development in box production technology?

5. Is CC autonomy at prototype level (that is during design stages) an essential feature needed for the design and development of complex systems (that is even for systems that don't feature CC autonomy)?

6. Which implications for control code production methods derive from an objective to create, to guarantee, or to preserve control code autonomy?[38]

[37] For microprocessors, however, the situation may be comparable with control codes.

[38] In [11] a collection of instruction sequences allowing jumps in to other instruction sequences within the collection is referred to as a polyadic instruction sequence (polinseq). Polyadic instruction sequencing refers to writing or designing polinseq's. This phrase is used instead of the common phrase programming in compliance with the view of [10] that programs represent polinseq's.

Compilation (with an emphasis on the code generation phase) is the process that automatically translates (projects in the terminology of [10]) a polinseq to a polyadic CC. Typically compilation is machine type dependent and it is

7. Is the emergence of CC autonomy an evolutionary phenomenon, in the sense that a technology making use of CC autonomy is more competitive than a technology without CC autonomy? Or can we merely say that, given the survival of technology lines that offer CC autonomy in each generation, CC autonomy is a sufficiently strong feature not to trigger fast elimination by other technologies that don't have it on offer.

8. If the latter is the case, is such a situation still in need of an explanation, or is it simply sufficient to point to today's IT marketplace and to assess that CC autonomy is still a common feature. In the latter case the question whether or not a killer technology for CC autonomy exits or can be developed is ignored and one is satisfied with the observation that such technologies have not yet been developed to a sufficient level of proliferation to threaten the survival of CC autonomy featuring product lines, without making any prediction about the future of CC autonomy as a technological strategy.

9. Even if CC autonomy is featured by equipment used for machine builders and designers that does not imply that CC autonomy is in any way visible for a larger user community. Is it conceivable that CC autonomy will become a niche phenomenon unknown to large user communities. This would apply already if all downloads of new CC's are performed automatically from the internet but without introducing new functionalities at least from a user perspective.

7.3 CC autonomy and the single box case

Assuming the presence of only a single box already introduces several scenarios that can serve as a business case for CC autonomy from a user's perspective. Here are three examples of such scenarios.

- *CC development and subsequent execution on the same machine.* User U may imagine a new functionality. When a polinsequensing environment is available on his machine he can develop the application himself on his machine. The CC is available as data. As a CC he can execute it on his machine, as data he can exchange it via the internet. In this setting it would be an unreasonably artificial constraint to maintain that U's machine is not code controlled. If so that is probably due to strict security mechanisms which actively block CC transportability into U's machine. In any case U works as if the CC's are independent.

- *Downloadable CC's for remote machines.* Imagine a bored machine user U on the moon unable to produce his own computer games. By remotely downloading a CC for a new computer game he can keep up to date with his children's progress on gaming. Possibilities for physical transportation of a CC are minimal, options for acquiring a new non-code controlled game console allowing to play the latest game are practically absent.

- *Downloadable CC's for new functionalities.* Imagine a user U who bought a laptop with internet connection. For U the concept of internet telephone might have been novel or unknown when buying his machine. But downloading Skype allows him to make unexpectedly

performed by a code controlled machine itself.

I propose the *polyadic instruction sequencing hypothesis* which states that a technology that features CC autonomy makes use of CC's which result from compilation of polinseq's. Further the design and development of codes takes place in terms of polinseq's rather than in terms of (executable) polyadic CC's.

The *high level polinseq hypothesis* adds to this that an evolution of CC's making use of the design of polinseq's will stagnate if only the bare notational format of [10, 12] may be used. Instead it is to be assumed that a systematic evolution of polinseq's can only take place if these are themselves found by way of compilation from higher level (that is more abstract or generic) notations (so-called high level program notations).

Taking this further it may hypothesized that different high level program notations are engaged in a competition between their owner/designers. This fierce competition is needed to make sure that the CC market place is so active that the additional costs of CC autonomy are compensated (for sufficiently many users) by its advantages.

cheap international connections with cellular phones throughout the world. So he is happy that this can be done by downloading a new CC (in this case from Skype).

8 CC autonomy considered a feature

The simplest conception of CC autonomy for a box type is that it constitutes a feature that the clients of its manufacturer or retailer have a substantial interest in. Like an open roof for a car, or a built in CD-player. It has been added because customers are willing to pay for it. In the case of CC autonomy one may wonder why users want to pay for it but that may be simply a matter of taste. Now continuing with the features of cars as an example, hybrid propulsion which may also be considered a feature of a single car is a comparable feature. Hybrid cars can be used in very much the same way as conventional cars with combustion engines only.

But electric cars (without combustion engine) are a wholly different matter. For a single electric car to exist there is a need for a full scale distribution system for electricity with dedicated charging stations. Thus electric propulsion is a feature that cannot be understood in isolation for a single car. Similarly lead free petrol consumption is a feature of car engines which may become dominant without any users being particularly enthusiastic about it, but its adoption requires that many car drivers participate in its use. We can formulate some assumptions about CC autonomy as a feature.

1. If boxes of type t are sold to a significant market and type t prescribes CC autonomy (that is boxes of type t are code controlled) than either this state of affairs is known to all users and these have obtained instructions on how to make use of it illustrated by convincing use cases or else this state of affairs is undisclosed to all users, and the option to exchange a major CC in a box B of type t may at best result from hacking B, or from covert operations from the side of the control code provider.

2. Most if not all boxes capable of delivering entertainment services provide CC autonomy.

3. Exploiting CC autonomy from the side of the user can have one or more of three major and different objectives:

   - enabling a box to establish new functionalities (perhaps by executing newly obtained CC's that were designed and manufactured after production of the box, and more importantly after the user's acquisition of the box),
   - improving the quality of a service (functionality) already present in B at the time of delivery to its customer (typical example an OS security update for a laptop), and
   - allowing a user to design, develop and execute a new functionality himself. This requires that box B is equipped with a complete programming environment. The latter feature was prominent for classical workstations (popular between say 1980 and 2000) of it is still present on today's PC's and laptops. But it has been omitted with cell phones and it has not been explicitly included into a number of smart phones that were developed on the basis of cell phones although these machines have functionalities quite comparable to a laptop.

4. The fact that a computer provides a full fledged instruction sequence production environment capable of manufacturing control code for itself may soon be a romantic memory from the past. It is difficult to see why that arrangement constitutes and advantage on the long run. As instruction sequencing becomes more involved, increasingly complex systems are needed for its support. For hardware design it is equally unconvincing that the design or the redesign of a hardware type can be performed with the exclusive help of a box of that type. At least for boxes that serve as components of safety critical systems it is plausible that the systems

used for their CC design and development are larger and faster than these boxes themselves. On the long run that will hold for most box types.

5. If type t specifies more or less fixed functionalities and the design of type t moves through a quick evolution mainly focused on improving the attractiveness of the functionalities offered (typical example: digital cameras), then improvements in control code are dealt with in the same way as hardware improvements (box improvements). If CC improvements constitute sufficient grounds for selling new boxes (with fixed CC's) then manufacturers obtain an advantage from not offering CC autonomy and from selling new boxes more frequently.

6. Assuming that CC distribution can be both quick and cheap a new CC (for instance for a game) can be developed into a hype which generates substantial profits. Indeed if there is a large basis of boxes of type t and a new service S for type t is developed as a CC (say C_p) then S can be distributed by merely distributing instances of C_p. This is a very important mechanism comparable to the development of top-hits in pop-music.

   Thus competitive development of CC's is made much more challenging and potentially rewarding when a large base of CC independent boxes is around. The gaming industry makes ample use of this mechanism. It is reasonable to believe that this will not change in the near future.

7. A further major incentive for CC autonomy is CC quality improvement during box life-time. Major examples are OS replacement and OS updates, including the very frequent security updates. All complex CC's may require regular updates in principle. This mechanism forces users into system administration and this unfortunate necessity is a disadvantage which simply reflects the unstable and poor quality of many CC's distributed today. In office networks centralized administration will often take these tasks out of the hands of PC (or laptop) users and by forbidding users to install CC's at their own initiative (not giving them system administration or superuser rights) the boxes are made non CC independent from a user's perspective.

9 Control code testing for safety critical embedded systems

After a survey of software testing literature Middelburg concludes in [31] that no formalization of software testing as an experimental process can be found in the computing literature and that no results are known about what information concerning computer software can be established better by means of testing than by other methods of investigation if such are available. I assume that these conclusions are valid for control code testing as well, because that can be considered a sub-theme of software testing. In this section I will argue that there is a comprehensible rationale for control code testing in the context of safety critical embedded systems. This rationale is not dependent on a hypothesized but unproven superiority of testing over other means of analysis according to whatever criterion.

I will now assume that a prospective user U intends to operate a robotic system R containing at least one code controlled machine M and that he contemplates the usage of polyadic control code C_p for a safety critical operation of R. In a practical context R will probably contain more computational devices each of which are likely to execute several control codes but in order to simplify the discussion these pluralities are ignored.

I assume that once started U will be unable to replace C_p if he concludes that it causes problems. Further I assume that failure of the operation of R is likely to cause bodily harm to some persons different from U. Thus R is a safety critical system. M inside R is an embedded computer. It is assumed that R's behavior consists of communicative signals and messages exchanged with its environment as incorporated in R. A plausible view is that M is in control of parts of the behavior of R and C_p is in control of the behavior of M. How activities of M are triggered and terminated is left unanalyzed.

9.1 Safety critical system components in detail

Suppose that U is completely unwilling to perform any theoretical analysis in advance in order to predict what happens when R is put into action. How can U arrive at the conclusion that R is safety critical? Only by actually putting R into action and then observing the bodily harm done, perhaps repeating this until he becomes convinced that the irreversible damage inflicted is probably caused by his putting R into action under certain conditions. This way of information gathering should not be included under testing. It constitutes a rather drastic form of learning by doing, by modifying designs containing mistakes after they have surfaced so until their causes can be identified and valid designs can be put in place.

If U insists to infer mission criticality of R without taking the risk that irreversible damage occurs he will need some predictive capacity which draws conclusions in the absence of actual operations. This implies that a minimum of theory is needed. Consider jumping out of an airplane equipped with a parachute. How to conclude that the parachute is a safety critical system component? This can be done in three steps: (i) conceptually replace the parachute by a device that will instantly dissolve in the air, (ii) then use the laws of gravity plus known facts about air resistance to compute an expected speed of ground impact given the totally dysfunctional parachute, (iii) use knowledge about the tolerances of the human body to infer that severe consequences are likely.

Thus a component of a system is safety critical within that system if it can be hypothetically replaced by another component in such a way that known theory makes the prediction of harmful consequences from a hypothetical operation very plausible. Returning to user U with his robotic system R containing embedded code controlled device M with polyadic control code C_p: U can conclude that M is safety critical within R if it can be hypothetically replaced by a similar device M' with different behavior which is very likely to cause severe damage when put into operation inside R. These conclusions must be based on some theory about the behavior of R depending on the functionality of M and on some know relation between problematic behavior of R and expected inflicted damage.

For this definition to make real sense U needs to know in addition that some M can be imagined which, when installed inside R will allow its adequate operation thus avoiding harmful consequences. If no such M is conceivable there is no basis for the assumption that M is a safety critical component of R because the design of R itself is essentially flawed. Obtaining this information is far from trivial. It requires a sound model of R and how its behavior depends on various variations of M from which predictions can be inferred. Nevertheless if M is to be considered a safety critical component of R this second piece of information must have been somehow established, potentially a highly demanding task requiring formal models simulations and so on.

This pattern can be repeated when considering C_p. This PCC is safety critical for component M inside R provided:

1. M is safety critical inside R,
2. For some conceivable C'_p the prediction that operating $R[M[C'_p]]$ will not cause severe damage can be made with sufficient confidence,
3. For some conceivable C''_p the prediction that operating $R[M[C''_p]]$ will have severe adverse consequences can be made with a very high level of confidence.

Each of these conditions can only be established by means of a theoretical account of the systems involved which allows making predictions. Of course that account may in part need to make use of experimental (that is measured) data, comparable with the parachute case above. Only if gravity, as measured, exceeds a certain threshold the feared negative consequences can be expected and so on.

9.2 Operating safety critical systems with safety critical components

It is assumed that at some moment U will indeed decide to start the operation of R. The situation for U is as follows:

- U must hope for the best.
- U has a certain amount of confidence that the mission will proceed without serious problems. This confidence is the primary justification of the decision to put R into action.
- U's confidence is primarily based on the confidence that none of the safety critical components of R (recursively) will cause failure of the operation of R. This decomposed confidence can have different grounds for different parts of R:
  - in some cases it is based on trust concerning the producer of a part manufactured elsewhere,
  - in other cases it is based on a full and formal validation of the design of system components in relation to the context in which these will operate,
  - yet in other cases new experimental data concerning system components will produce information needed to complete overall validation.

Anyhow the prediction that R will work without inflicting damage cannot be produced by mere logical reasoning. Some knowledge about the system and its components must have been collected by other means. Altogether the ultimate category that guides U's decisions is confidence rather than logical proof.

9.3 Plausibility of control code testing for safety critical control codes

Given the definition of safety criticality for C_p as outlined above: when will U consider testing of C_p a an activity which increases his confidence that operating R will not crash on problems caused by C_p. This is a matter of hypothetical psychology rather than logic. Nevertheless the following sequence of observations leads to a confirmation that control code testing can be a rational way of proceeding for which no obvious alternative can be put forward:

- If U knows that C_p has been produced via instruction sequencing he will prefer to have an analytical execution architecture of $\mathrm{M}[\mathrm{C}_p]$ on which to base the prediction that this component will not cause any problems within R. This kind of analysis can serve a role as well for validating the second condition of safety criticality before arriving to the conclusion that C_p is indeed safety critical (for R as deployed in M).
- Tests consisting of operations of $\mathrm{R}[\mathrm{M}[\mathrm{C}_p]]$ in safe circumstances (that is circumstances where failures of R do not have the adverse consequences feared in actual use) where no problems are reported which can be traced back to C_p will increase confidence of U in C_p.
- A test with a modified PCC, C'_p of a system variation $\mathrm{R}[\mathrm{M}[\mathrm{C}'_p]]$ that develops a failure not present in a test of $\mathrm{R}[\mathrm{M}[\mathrm{C}_p]]$ produces important belief that C_p is safety critical, thus creating justification for a thorough validation of the second criterion that some conceivable PCC is capable of controlling M in a desired way, and additionally increasing the justification for extensive testing of R with circumstances that require adequate operation of M.
- From this observation it may be inferred that control code testing should include testing of systems that execute control code inside embedded devices. Nevertheless it seems reasonable to distinguish pure control code testing where the control code is considered a part of a computing device only from embedded control code testing where the embedding of the code controlled device is also taken into consideration.

- The more known the behavior that M is supposed to produce is, the more plausible it is for U to derive confidence in C_p from trust in its manufacturer.

- When U is in need to make a best possible attempt for operating R under time pressure because a delay of operating R may lead to damage comparable to the damage expected from its faulty operation, it can be reasonable to base confidence in the safety critical control code C_p on a series of successful tests in combination with an informal analysis of an analytic execution architecture of $M[C_p]$.

- Tests become more informative as precise specifications of the behavior expected from M and precise information concerning the interface between M and R is lacking. It such circumstances tests reduce uncertainties about the context of C_p rather than uncertainties about C_p itself. These uncertainties cannot be reduced by merely considering $M[C_p]$ and validating (by means of testing or by means of formal analysis or a combination of these) that under control of C_p the device $M[C_p]$ works according to some given specification.

- In the latter circumstances control code testing cannot be replaced by a formalized alternative means of analysis even if the polyadic control code is known to be a compiled polyadic instruction sequence and an analytic execution architecture for M is known that enables predicting its behavior for a given C_p.

- It seems to rest on some form of general engineering intuition to propose and believe that some C_p' can be found in principle such that $R[M[C_p']]$ will not fail. This is very much connected with a notion of universality of M. Once M is accepted as universal (a matter of trust) and taking a fairly philosophical perspective on the architecture of R it follows somehow 'logically' that an adequate control code C_p' exists in principle; it just needs to be found.

  Remarkably the combination of this philosophical perspective with the assumption of M's universality allows to infer the positive information needed to assert that C_p is safety critical without much formal modeling work. This is consistent with an informal view of the interaction between M and R and it creates a context where pure control code testing (which might in principle be replaced by alternative methods of analysis for control codes explained by an analytic execution architecture) must be augmented with embedded control code testing (which cannot be replaced by generic formal methods from computer system semantics).

- So I conclude that an intuition of universality of control coded devices plus a general perspective on what may be expected from an embedded digital device creates a setting where control code testing and its embedded extension can play a vital role in the design of safety critical systems, a role that cannot easily be replaced by formal analysis or verification, in particular not in the presence of temporal pressure.

- This conclusion stands even if one agrees that the question raised by Middelburg in [31] about when pure software testing (here identified with control code testing given that control codes have been produced by means of instruction sequencing) is more effective or efficient than other means of analysis (which are guaranteed to exist on the basis of an analytic execution architecture) not involving the physical process of control code execution, is still open (and for that reason might have a negative answer).

References

[1] W.P. Alston. The quest for meanings. *Mind*, 72 (285):79–87, 1963.

[2] C.E. Althaus. A disciplinary perspective on the epistemological status of risk. *Risk Analysis*, 25(3):567–588, 2005.

[3] P. Ammann and J. Offutt. *An introduction to software testing*. Cambridge University Press, 2008.

[4] A. W. Appel. Axiomatic bootstrapping: a guide for compiler hackers. *ACM TOPLAS*, 16(6):1699–1719, 1994.

[5] G.A. Avlonis. Product elimination decision: does formality matter? *Journal of Marketing*, 49:41–52, 1985.

[6] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold 2nd edition, 1990.

[7] J.A. Bergstra. Formaleuros, formalcoins and virtual monies. 2010. `arXiv:1008.0616 [cs.CY]`.

[8] J.A. Bergstra and I. Bethke. Predictable and reliable program code: virtual machine based projection semantics. In J. A. Bergstra and M. Burgess, editors, *Handbook of Network and System Administration*, pages 653–686. Elsevier, Amsterdam, 2007.

[9] J. A. Bergstra and M. Burgess, editors. *Handbook of Network and System Administration*, Amsterdam, 2007. Elsevier.

[10] J. A. Bergstra and M. E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002.

[11] J.A. Bergstra and C.A. Middelburg. Thead extraction for polyadic instruction sequences. 2008. `arXiv:0802.1578 [cs.PL]`.

[12] J.A. Bergstra and C.A. Middelburg. Instruction sequence processing operators. 2009. `arXiv:0910.5564 [cs.PL]`.

[13] J.A. Bergstra and C.A. Middelburg. Machine structure oriented control code logic. *Acta Informatica*, 5(1):170–192, 2009. `arXiv:0711.0836 [cs.SE]`.

[14] J.A. Bergstra and A. Ponse. Execution architectures for program algebra. *Journal of Applied Logic*, 46:375–401, 2009.

[15] J.A. Bergstra and A. Ponse. A progression ring for interfaces of instruction sequences, threads and services. 2009. `arXiv:0909.2939 [cs.LO]`.

[16] J.A. Bergstra and P. Walters. Operator programs and operator processes. *Information and Software Technology*, 45:681–689, 2003.

[17] M. Colyvan. Is probability the only coherent approach to uncertainty. *Risk Analysis*, 28(3):645–652, 2008.

[18] R.M. Cyert, H.A. Simon, and D.B. Trow. Observation of a business decision. *The Journal of Business*, 29 (4):237–248, 1956.

[19] J. Earley and H. Sturgis. A formalism for translator interactions. *CACM*, 13(10):607–617, 1970.

[20] V.A. Gibson and M. Louargand. Risk management and the corporate real estate portfolio. *American Real Estate Society Annual Meeting*, pages 1–17, 2002.

[21] T. Haavelmo. The notion of involuntary economic decisions. *IEconometrica*, 18 (1):1–8, 1950.

[22] M. I. Halpern. Machine independence: Its technology and economics. *CACM*, 8(12):782–785, 1965.

[23] S.O. Hansson. Fallacies of risk. *Journal of Risk Research*, 7(3):353–360, 2004.

[24] T. Horlick-Jones. Informal logics of risk: contingency and modes of practical reasoning. *Journal of Risk Research*, 8(3):253–273, 2005.

[25] L.E. Janlert. Dark programming and the case for rationality of programs. *Journal of Applied Logic*, 6(4):545–552, 2008.

[26] R. Jervis. Political decision making: recent contributions. *Political Psychology*, 2 (2):86–101, 1980.

[27] H. Jungermann. Speculations about decision-theoretic aids for personal decision making. *Acta Psychologica*, 45:7–34, 1980.

[28] P. Kruchten. A rational development process. *Crosstalk*, 9(7):11–16, 1996.

[29] D. Mahalel, D. Zaidel, and T. Klein. Driver's decision process on the termination of green light. *Accident analysis and prevention*, 17 (5):373–380, 1985.

[30] C.A. Middelburg. Searching publications on operating systems. 2010. `arXiv:1003.5525 [cs.OS]`.

[31] C.A. Middelburg. Searching publications on software testing. 2010. `arXiv:1008.2647 [cs.SE]`.

[32] J.M. Moritz. Rendering unto science and God: Is NOMA enough? *Theology and Science*, 7(4):363–378, 2009.

[33] M. Peterson. The logical status of risk-to burnish or to dull. *Risk Analysis*, 26(3):595–601, 2006.

[34] E.A. Rosa. Metatheoretical fundations for post-normal risk. *Journal of Risk Research*, 1(1):15–44, 1998.

[35] E.A. Rosa. The logical status of risk-to burnish or to dull. *Journal of Risk Research*, 13(3):239–253, 2010.

[36] H.A. Simon. Theories of decision-making in economics and behavioral science. *The American economic Review*, 49 (3):253–283, 1959.

[37] R.L. Simons. A note on identifying strategic risk. *Harvard Business School Publishing, Boston*, 1999.

[38] M. Stenmark. A religious partisan science? Islamic and Christian perspectives. *Theology and Science*, 3(1):23–38, 2005.

[39] Y. Wand, D.E. Monachi, J. Parsons, and C.C. Woo. Theoretical foundations for conceptual modelling in information systems development. *Decision Support Systems*, 15:285–304, 1995.