Transmathematica DOI Prefix 10.36285 ISSN 2632-9212

Accusations in the Context of Computer Programming

Jan A. Bergstra
j.a.bergstra@uva.nl, janaldertb@gmail.com
Informatics Institute, University of Amsterdam,
Science Park 904, 1098 XH, Amsterdam, The Netherlands

Marcus Düwell
mduwell62@posteo.net
Institute for Philosophy, Technical University of Darmstadt, Germany

Submitted: 31 January 2022
Revised: 3 October 2022

Abstract

We imagine a minimal context where a programmer A serves as the provider of a program X for a user B. A survey is given of plausible promises and accusations in connection with the delivery and deployment of X. When X is used, various problems may arise. The focus of the paper is on the role of accusations which may plausibly arise in various scenarios. We discuss different accusation patterns that may occur outside informatics and see whether such patterns may appear in the context of programming as well.

1 Introduction

Writing about promises, threats, and accusations in the context of instruction sequence theory provides two characteristics which we wish to exploit: (i) computer programming provides many examples for promises and threats, and (ii) transactions between a programmer and a user provide ample opportunity for making accusations by either side.

In what follows, a program will be understood as a digital entity which qualifies as a program according to the definition of (computer) programs given in [8]. In brief the definition states that a program is an entity which obtains its meaning by way of a translation to a repeating single pass instruction sequence in the program algebra notation of [12, 14] and [13].

1.1 For informatics: promise of fact is prior to fact

In promise theory as proposed by Mark Burgess (e.g. [16]), the messaging of a fact acquires the status of a promise. This may sound counterintuitive but the argument is convincing. Consider a database D_{db} which contains the information that "the address of person P_{person} is $a_{address}$. On request of agent B the database may now issue promise p with promiser D_{db}, promisee B_{person}, content (also body): $\texttt{address-of}(P_{person}, a_{address})$. and with D_{db} and B_{person} in scope.

From the perspective of D_{db} the assertion $\phi \equiv \texttt{address-of}(P_{person}, a_{address})$ is epistemic in the sense that this response is at best in accordance with the most recent information available to D_{db}. Whether or not ϕ is actually true is not known to D_{db}, and is not at issue for D_{db}.

In [24] the epistemic modality (fact according to some agent), is distinguished from objective modality, and objective modality ramifies into a spectrum of modalities among which metaphysical modality and nomic modality. The latter two modalities may be explained in terms of necessity or possibility. Suppose one understands ϕ as the necessity that (at present) $\texttt{address-of}(P_{person}, a_{address})$, or stated differently the negation of the possibility of $\neg\phi$. Metaphysical modality is about the principled imagination that $\neg\phi$ would be valid, whereas nomic modality is about the compossibility of $\neg\phi$ with the laws of nature. Suppose it is known that updates reach D_{db} with a certain speed then at some instant of time it may be the case that $\neg\phi$ is metaphysically possible (the update of a new address for P_{person} may not yet have reached the database, while $\neg\phi$ is nomically impossible because the update mechanism works in such a manner that an update must have been processed at the moment promise p is being made, and in addition it is known that the change of address of P_{person} cannot have been very recent.

For informatics, at least as far as its application is concerned, promise serves as a modality which is agnostic of underlying modalities. For an agent C_1 in scope of D_{db} the promises that it produces may be merely epistemic, that is, they may be more informative about the database itself than about the person about whom information is sought. For C_2 there may always be the metaphysical option, to doubt whatever might be doubtful, whereas for C_3 the same database may be just a piece of laboratory equipment which primarily is to be understood by way of the laws of nature. In summary:

Claim 1.1. *(Factual promise priority.) Taking computed output as a promise of fact is prior to taking it as a fact.*

When contemplating promises about programs the three modalities: epistemic, metaphysical, and nomical each have a role to play. However, in general a promisee may not know which of the modalities (if any) a promiser has in mind when making a promise.

1.2 Promises in connection with programs

We will work with a rudimentary agent model: a programmer, say A, manufactures a program for a user, say B (representing a user community), and some agents in scope of the process, (for example a common manager $M_{a,b}$ of A and B, and/or some certification authority C, or a representative member of the user community, and/or a lawyer known to A and B). The basic promise in programming is that a programmer feels that a job has been done.

1.2.1 Primary promises: program checkout and program checkin

Two promises, that is promise types, will be considered primary promises in connection with programming, program checkout promises and program checkin promises.

Program checkout promise: (programmer) A promises to (customer/user) B (with scope U) that program X provides a satisfactory solution for the problem captured in requirements S_{req}.

For the use of checkout in programming see e.g. [18]. Checkout may take place on the basis of a programmer's subjective assessment that the job has been "done". In the latter case the promise uses an epistemic modality. However, it may also be the case that the programmer has proven beyond any doubt that

X correctly implements S_{req} in which case the promiser may claim metaphysical necessity of the claim that "the work is done". Alternatively in cases where the program interacts with physical reality (for example the control of an embedded system in an airplane, or the control of a robot) nomical necessity of the correctness of the system may be what A intends to communicate.

Claim 1.2. *(Checkout promise priority.) The program checkout promise is prior to any agent's awareness of the modality.*

Indeterminacy of the modality of the promise body will be present in most promises and for that reason it will not be mentioned explicitly below. Acceptance is primarily a matter for the customer/user of a program. In [22] it is made explicit that acceptance comes with a customer side perspective and it is also emphasised that the notions involved are quite informal. Acceptance is complementary to checkout. In view of this complementarity we uses checkin promise rather than acceptance promise.

Program checkin promise: (customer/user) A promises to (programmer) B (with scope U) that program X provides a satisfactory solution for the problem captured in requirements specification S_{req}.

The program checkin promise may also be referred to as program acceptance promise. The latter phrase comes with a disadvantage, however, that acceptance testing is often considered a task for the software developer. Checkin does not give rise to such confusion. The program checkout promise does not require a specific methodological underpinning. We use "delivery" if some model for software quality assurance is available and is being followed.

Program delivery promise: (programmer) A promises to (customer/user) B (with scope S) that program X provides a satisfactory solution for the problem captured in requirements specification S_{req} and that its delivery in the given state is in accordance with specified guidelines G to that extent.

1.2.2 Secondary promises for programming: requirements specification and technical specification

Prior to an exchange of X the programmer and the customer/user are likely to exchange a requirements specification S_{req} and a technical specification S_{tech}.

Requirements proposal promise: (customer/user) B promises to (programmer) B (with scope U) that their needs are captured in requirements specification S_{req}.

Requirements acceptance promise: (programmer) A promises to (customer/user) B (with scope U) that their needs as captured in requirements specification S_{req} constitute an adequate starting point for designing a corresponding technical specification S_{tech}.

Technical specification checkout promise: (programmer) A promises to (customer/user) B (with scope U) that their needs as captured in requirements specification S_{req} are adequately dealt with in technical specification S_{tech}.

Technical specification acceptance promise: (customer/user) B promises to (programmer) B (with scope U) that their needs as captured in requirements specification S_{req} are adequately dealt with in technical specification S_{tech} as proposed by A.

Many more promises can be imagined in connection with computer programming and with the use of computers.

1.3 Promise: an unnecessarily complicated modality?

Promise theory according to [9] views a promise, say p, as an entity under some description which originates from a promiser, say agent A, is directed towards a promisee, say agent B which has a body β containing the content of p an a scope, say S which is a collection of agents at least including A.

Promise theory takes the notion of a promise as a primitive. One may understand the notion of a promise as a relation having at least two agents and a proposition, that is the body, as arguments. The fact that β has been promised may be understood as a modality, though without any assumptions about an underlying logic. Promises involve epistemic modality, and are remote from objective modalities (see for example [24] for an account of a distinction between objective modality and epistemic modality). We consider the following notations:

- $\boxplus_A^{B,S}\beta$: it is the case that A promised (with scope S) β to B,
- $p\,\boxplus_A^{B,S}\beta$: a named promise: $\boxplus_A^{B,S}\beta$ and this promise carries name p,
- $\boxplus_A^{B,S}+\beta$: it is the case that A promised (with scope S) to offer service β to B, (named version $p\,\boxplus_A^{B,S}+\beta$),
- $\boxplus_A^{B,S}-\beta$: it is the case that A promised (with scope S) to accept service β from B (named version $p\,\boxplus_A^{B,S}-\beta$),

If there is a plausible notion of obligation, say $O_A\beta$ for A is obliged to see to it that β, then promising will not create an obligation for the promiser:

$$\Diamond(\boxplus_A^{B,S}\beta \wedge \neg O_A\beta)$$

Promising may have a side effect on trust (see also Paragraph 1.7 below) which is in part determined by the promise being kept (in the perception of the promiser). Promising may also have a side effect on expectations which is in part determined by a level of trust.

- $\boxplus_A^{B,S}\beta \wedge trust_B^{>a}A \rightarrow expect_B^{>b}\phi$: if B has significant (say more than a) trust in A and A promises β to B then B will have a significant (say more than b) expectation that ϕ will be true or will become true.
- $\boxplus_A^{B,S}\beta \wedge trust_B^{<u}A \rightarrow expect_B^{<v}(E_A\beta)$: if B has low (say less than u) trust in A and A promises β to B than B will have a significant (say less than b) expectation that ϕ will become true due to A. Here $E_a\phi$ expresses that A causes ϕ to be true (following [23]).

We do not claim that a convincing modal logic of promises exists or can be found. The idea however is that the primary effect of promising consists of the impact that a promise has on the expectations of the promisee

as well as the expectations of the agents in scope. Said impact may vary in time in various ways. We assume the following laws of the dynamics of expectations.

From a logical perspective choosing a relation of such complexity as a primitive requires a motivation. In some cases one can imagine that β is unique in time and space, which is characterised by a channel c via which β is communicated:

$\boxplus_A^{B,S} c\colon \beta$: it is the case that A promised (along channel c and with scope S) β to B. Now one may assume:

$$\boxplus_A^{B,S}\colon \beta \iff \exists c[\boxplus_A^{B,S} c\colon \beta]$$

moreover, assuming that c allows the communication of a unique promise only one may contemplate that a conjunction of two simpler modalities, both involving fewer agents as arguments, suffices to explain the notion of promise:

$$\boxplus_A^{B,S} c\colon \beta \iff (\exists a[\boxplus_a^{B,S} c\colon \beta]) \wedge (\exists b[\boxplus_B^{b,S} c\colon \beta])$$

Now two simplified modalities can be imagined:

$$\boxplus_{\bigstar}^{B,S} c\colon \beta \equiv \exists a[\boxplus_a^{B,S} c\colon \beta] \text{ and } \boxplus_A^{\bigstar,S} c\colon \beta \equiv \exists a[\boxplus_a^{B,S} c\colon \beta]$$

both with fewer agent arguments, so that

$$\boxplus_A^{B,S} c\colon \beta \iff \boxplus_{\bigstar}^{B,S} c\colon \beta \wedge \boxplus_A^{\bigstar,S} c\colon \beta$$

However, the idea that promising can be expressed as a conjunction of simpler modalities is unwarranted and the program checkout promise provides an example to the contrary: consider an instance of the program checkout promise

$$p \boxplus_A^{B,U} [\texttt{program}\ X\ \texttt{implements}\ S_{req}]$$

Promise p is unsurprising (say for agent $C \in U$) only if (C believes that) A knows for what purpose B will use X, (unless C believes that A intends to deceive B).

In fact promise p can only be properly justified under a bundle of additional preconditions: (i) that A understands S_{req} (more often than not an informal document), (ii) that A understands what, according to B it means that S_{req} is implemented by X, (iii) that B understands what kind of credibility this kind of promise from A can be assigned, (iv) that A is aware of the implications for agents in scope S if they adopt promise p by expecting that X will be used by B for its intended objectives.

1.4 Promise versus obligation

Promise Theory (PT) aims for a formal understanding of promising without any assumption of the creation of obligations by consequence of an act of promising. It is hypothesised that promising is a tool for voluntary cooperation in the absence of side effects regarding obligations, in a context of animate agents as well as in a context of inanimate agents. It is important to understand that PT is primarily an approach for the investigation of the role of promises in communication and particularly in the context of computer programming. PT does in that sense not strive for a comprehensive account of promises which would require much broader discussions in moral and social philosophy. Perhaps the analogy with Kant's distinction between philosophy of law and virtue ethics is informative for understanding the aim of PT. In the context of the law the role of promises can be described without referring to the concrete psychology and the motivational structure they may have for human agents. A virtue ethics, however, would have to understand in a more comprehensive way how promises are embedded in the self-understanding of agents. The latter theme may be vacuous for certain classes of non-human (for instance artificial) agents. In a human context an account

of promises will raise a lot of questions regarding the relationship between promising and a sense of obligation. It is important to see here, however, that promiser, promisee, and agents in scope of a promise may entertain entirely different views on the quality, form and persistence of obligations which supposedly arise from promising. PT does not aim to give definitive answers to those questions but provides a conceptual framework that facilitates a nuanced way to discuss those questions. In the case of computer programming there is ample room for promises which create and interact with expectations rather than obligations.

1.5 Threats as counterparts to promises

A threat is a negative counterpart to a promise. Threats play a significant role in computing, in particular in the context of security threats, but less so in computer programming. Threats are discussed in more detail in [5] and the reference cited there (see also [2] for the concept of threat). We suggest the following notations: $p\boxminus_A^{B,U}\beta$ for a named threat (name p) issued by A to B with scope S (containing A, not necessarily containing B) with body β, and $\boxminus_A^{B,U}\beta$ for the corresponding unnamed threat.

A threat in programming may occur if programmer A threatens customer/user B to stop working on program X (which B expects to be developed by A and for B about halfway next year.

1.6 More secondary promises in relation to computer programming

For completeness, but without relevance for the rest of the paper we list some more promises of relevance for programming. Scope is left implicit. We qualify these promises as secondary because each of these plays a role in processes meant to arrive at a stage in which both primary promises, that is the program checkout promise and the program checkin promise, have been made.

1. B promises A that offers for work on a project for implementing requirements specification S_{req} are welcome, and that such offers will be considered in detail
2. A promises B to turn S_{req} into a technical specification S_{tech}.
3. B promises A payment if a program X is developed so that X $\underline{\texttt{sat}}$ S_{tech}.
4. A promises B to develop a program X so that X $\underline{\texttt{sat}}$ S_{tech} (that is an implementation of S_{tech}),
5. A promises B to act quickly on bug reports by delivering a patch (that is a modification X' of X provided by A), or a mechanism available to B for it it to modify X into X'.
6. B promises A (by way of bug report number k) that (with a run of X) on a particular input p_1 the output $q_1 = X(p_1)$ was observed which lies outside the technical specification S_{spec}, i.e. $\neg S(p_1, q_1)$ (and for that reason X is expected to fail on test case p_1 when tried out by A).
7. B promises A that no damage has been incurred (by B or a client of B) which is being considered by B as having been caused by the failure mentioned to B in bug report k.
8. B promises A (by way of bug report no. $k+l$) that on input p_2 the result $q_2 = X(p_2)$ of X lies outside the specification S, that is $\neg S_{spec}(p_2, q_2)$. Moreover B notifies A that a causal connection between this failure of X with damage that has recently occurred is investigated.
9. B promises A (with reference to bug report no. $k+l$) that, as was stated already, X failed on input p_2 as $\neg S_{spec}(p_2, X(q_2))$ and moreover, B confirms that a causal connection with damage D is considered plausible at this stage.

10. A promises B to investigate bug report number $k + l$ without delay and to report on the findings including providing answers to the following questions:

    (i) Is the bug recognised (that is can it be repeated on the development platform used by A), if not, is R_{req} based on invalid assumptions about the platform on which B is using X?

    (ii) Is the bug caused by a fault, if so, which mistake (made by whom) has caused the fault, if not has a software process flaw occurred?

    (iii) Why was the bug not detected when the program delivery promise was made,

1.7 The role of trust

The dynamics of promising is best understood under the assumption that the agents involved maintain, and regularly update, levels of trust with respect to other agents. We understand trust as follows:

Definition 1.1. *Trust of agent A in agent B is a willingness of A to accept vulnerability (following [1]) with respect to B, where this willingness is grounded in two beliefs: (i) A's adoption of some rule which governs the behaviour of B, plus (ii) A's expectation that B will operate in compliance with said rule (for both beliefs in the context of trust see [19]).*

Assuming that B (i) B is willing to accept vulnerability arising from failure of X, and (ii) B has strong belief in A's discipline and competence of program production (that is programming), and (iii) B believes that A will work at the usual quality level on this occasion, then, upon becoming aware of A's program checkout promise concerning program X, then B may become confident that X will work well for B (that is B will expect X to function in a satisfactory manner), and subsequently B may issue a complementary program checkin promise (for X) without hesitation or delay. Upon having made a checkin promise it is plausible that B makes use of X. Successful use will increase B's expectation that X is satisfactory, as well as B's belief (i) (as in Definition 1.1), thereby increasing B's trust in A as a programmer.

If no program is delivered by A, or the delivered program X does not work properly (according to B), B's expectation that X is satisfactory will decrease and B's trust in A may be negatively affected. In the latter case it will depend on the circumstances which of both beliefs (as in Definition 1.1) will be most affected.

2 Accusations

We will use the approach to accusations of [11]. The format is that, with accusation p, (accuser) A, accuses (accusee) B (with scope S) of β (body of the accusation). Accusations are similar constructs to promises but with quite different meanings. The following modal notations may be used: $\boxtimes_A^{B,S}\beta$ for an unnamed accusation and $p\,\boxtimes_A^{B,S}\,\beta$ for a named accusation, but we will make limited use of these notations below.

We notice that accusation theory has no bias in favour or against either the accuser or the accusee or in connection with any agent in scope. It seems to us that accusations occur in in practice in certain patterns and that a fairly limited collection of patterns describes a majority of the cases. In order to get a grip on the plurality of patterns we start with surveys of five aspects: (i) why are accusations made, (ii) what kinds of issue (subject of the body of the accusation) can be distinguished, (iii) which assessments of the strength of the accusable can be mentioned, (iv) to what extent is the accusation personal, and (v) which reactions to accusations can be distinguished.

2.1 Procusations as counterparts to accusations

For reasons of symmetry we introduce procusations as a sign of support to a procussee, the opposite to an accusation. Both accusations and procusations may be considered speech acts in which a judgement is issued about the behaviour of an other agent, and this is done with zero or more other agents in scope. An accuser asserts a negative attitude towards some feature (past, present or future) of the accusee, while a procuser asserts a positive attitude with respect to a feature of the procusee. If a restaurant receives a negative review which details the negative reviewer experience, such a review may be understood as an accusation, and conversely, if the review is positive such may be understood as a procusation. Procusations are promises as understood in promise theory, though with the additional restriction that a procusation will not suggest any future action by a procuser. For instance: B (customer, user) procuses to A (programmer) that program X has passed the acceptances tests and is ready for deployment by B. Below we will use promise also in cases where procusation might be preferable in order to keep the vocabulary as simple as possible. As notations for procusations we propose: $\boxdot_A^{B,S}\beta$ for an unnamed procusation and $p\,\boxdot_A^{B,S}\,\beta$ for a named procusation.

2.2 Why are accusations made

Given accusation p: A (accuser) accuses B (accusee) of β (body, that is content of p) with scope S (collection of agents at least including A). The general question may be posed: why might this happen? The idea is not so much to provide an exhaustive answer of this question but to uncover patterns with high frequency.

1. A feels that agents in S should know about B's involvement in β, and A packages this information in an accusation; this motive may or may not be combined with one or more of the following motives:

   (i) A intends and expects to achieve that A will be punished for its share in β,

   (ii) A intends to harm the status/position of B (including: A (primarily) acts out of revenge towards B, and: A takes pleasure in the resulting complications),

   (iii) A hopes to reduce the influence of B, in a certain context that A shares with B, (for instance by reducing the trust that agents in S have in B, or by forcing B into a defensive position),

   (iv) A expects to gain recognition and visibility among agents in S (and perhaps beyond S),

2. A holds that B must change habits, and by issuing the accusation p expects to contribute to that change coming about),

3. A expects some form of compensation to be awarded to agent C (maybe $C = A$) who A considers to be " victim of p", (either directly from B, assuming B is in S, or indirectly, that is from other agents in S),

4. A has in mind that agents in S come to know that the behaviour of B is considered "accusable", and, optionally, one or more of the following:

   (i) A indicates to agents in S to take notice of the fate of B upon having been accused in the manner of p,

   (ii) A indicates to agents in S to take notice of the fate of B upon having been involved in β,

   (iii) B is not alive anymore and cannot be an agent in S for that reason, nevertheless agents in S must take notice of the assessment of β and grasp that having been supportive of B may still have consequences.

5. other motives (for A).

2.3 What kind of wrongdoing is β according to the accuser?

An exhaustive listing is impossible but some forms of wrongdoing (in the eyes of the accuser) may be distinguished:

1. problematic personal behaviour (including: supposedly siding with the wrong side in a conflict, intentionally causing harm or damage; helping one or more agents with performing objectionable behaviour),
2. problematic unprofessional behaviour, (including: sloppy work or negligence; acting against codes of conduct of one's profession, not necessarily in a consequential way; poor quality work),
3. problematic inaction (including the following: having been inactive in a situation where one or more agents $C, C_1, C_2, ..$ were damaged while B could have been effectively helpful to them; not speaking out or otherwise intervening in a non-violent manner upon having become aware that an agent, say D runs the risk of being treated badly; not taking action in order to prevent certain risks from coming into existence or being mitigated),
4. other accusables.

2.4 Measuring the strength of accusations

We propose 5 levels of force of an accusation. These levels are independent of the validity of the accusation:

1. controversial,
2. marginal,
3. moderate,
4. significant,
5. serious.

2.5 What reactions to an accusation may be distinguished

Suppose A accuses B of β with scope S

- B considers the accusation valid and B promises to A (B's intention) to engage in a constructive debate on "what next?",
- B considers the accusation justified, while being as yet unconvinced of the validity of β, and B promises to A (B's intention) to engage in a constructive debate on

  - what next in terms of fact finding on β, and,

  - how to arrive at a joint assessment of β, and

  - what to do after the facts have been found, and agreement of an assessment has been achieved.

- B denies the accusation (that is B considers p both justified and invalid) and B promises to provide A with a motivated denial of (its involvement in) β,

- B considers the accusation unjustified and B retaliates, by means of one or more of the following:

  - B issues a counter accusation towards A, (perhaps with a different scope, perhaps with A not in scope),

  - B issues a threat of some kind towards A,

  - B plans to damage A and does so, or,

  - B asks agents in S to support B in harming A.

- other forms of reaction to an accusation.

2.6 Levels of abstraction and transformation of accusations

Given accusation p: A accuses B of β with scope S, then two important aspects of β are: compression and emphasis. Compression (degree of being comprimed) is about the level of detail of the description of β while emphasis has to do with the way in which components of β are being highlighted, possibly at the expense of other components of it. Given p one may imagine:

to comprime p : making β more detailed and extensive (alternatively to make p more abstract, to make it less concrete, to shorten p, to unrefine p, to make p more compact, and to compactify p),

to uncomprime p : making p more detailed (to refine p, to make p less abstract),

to emphasize p : with respect to an intended profile: β is understood as being a combination of components which according to a profiling policy have differential importance and for components of higher importance a presentation with more emphasis is chosen while for components with a lower importance a presentation with less emphasis may be chosen.

to de-emphasize p : with respect to an intended profile: β is understood as being a combination of components which according to a profiling policy have differential importance and for components of higher importance a presentation with less emphasis is chosen while for components with a lower importance a presentation with more emphasis may be chosen.

For instance take β (with B accusing A, however) as follows:

> program X has been delivered by A (programmer) to B (user) while there were more than 250 faults present in X, including 3 faults (as detailed in report R that C (consultant) has recently written on request of A) which together have recently caused substantial damage (as documented in d). Moreover A has not responded to bug reports from staff member b of B who indicated failures which were caused by one of the faults mentioned in R.

A may be unimpressed because (i) A holds that a presence of only 250 faults in the first release of X would in fact be a greats success (for A), and (ii) B's methods of fault localisation do not spot one of the 3 mentioned faults as causes of the failures which staff members of B have been reporting since its delivery to and deployment by B. Other faults were recently found (in search of causes for the allegedly neglected user bug reports, which are not causes of substantial damage as reported in d according to B.

Now β can be made more compact (for instance by C who is reporting on p for certain news outlets) as follows:

> program X has been delivered by A (programmer) to B (user) while there were many faults present in X, including faults which have recently caused substantial damage. Moreover A has not responded to bug reports from staff of B.

And upon emphasising C may obtain:

> programmer A has sold B buggy programs that caused much damage. Staff of B has responded poorly to complaints from B.

It is plausible that B is unhappy with the compactified form of p and even more unhappy with the emphasised form of the latter version of p.

3 Accusation theory as complementary to promise theory

Although accusation theory is mean to be able to stand on its own feet, without relying on promise theory, it is helpful to focus on the potential complementarity between both accounts. Following the ideas of Mark Burgess promise theory does away with the conventional side effect of a promise that it creates an obligation for the promiser, and thereby promises are turned into a tool for an underlying normative theory of voluntary cooperation. Similarly accusations can be set free from an assumption that by making an accusation the accuser acquires an obligation to demonstrate the validity of what we call the body of the accusation (using the corresponding terminology of promise theory). Now accusations can be understood as a tool for an underlying (equally normative?) theory of voluntary non-cooperation. Voluntary non-cooperation may be understood as a mechanism which is helpful for an agent to find out with whom to cooperate, which matters in turn because cooperation becomes more effective if it occurs in a clustered fashion.

Besides accusation there are several other mechanisms of voluntary non-cooperation. Simply lack of interest or lack of sympathy may incentivise voluntary non-cooperation without any accusation coming into play. The above suggestion amounts to no more than that accusation is one of the instruments of voluntary non-cooperation, where we notice that in a context where accusations are made in a very public manner these effects may be quite strong. For instance a high profile accusation may work well to change the loyalties of voters in advance of elections

3.1 The OS movement as an instance of voluntary non-cooperation

Voluntary non-cooperation plays a significant role in computer software. For instance the open source (OS) movement accuses companies of keeping software secret so that problems cannot be spotted and improvements cannot be proposed let alone implemented by competent members of the public. The mechanism of OS software implements voluntary non-cooperation with agents who entertain certain business models. A paradigmatic accusation (pattern) arises: A (proponent of OS) accuses (p), B (programmer) of selling C (user) a program X which (i) has been derived from OS program Y (with reference to Y) and either (ii) without making due reference to Y and without declaring that X has an OS status as well, or otherwise (iii) while making X closed source and imposing corresponding restrictions on the use by C of inSq X. OS uses copyright law to realise a context where accusations of the kind mentioned can not only be upheld in court but may also damage the reputation of B so much as to constitute an incentive for C (or any other potential user of X) not to cooperate with B in this manner.

3.2 The anti-testing movement as an instance of voluntary non-cooperation

It seems fair to say that within computer science there has been an attitude among some researchers to view a focus on program testing as a sign of bad taste. The underlying accusation would be that: those who focus on testing do not even try to contribute to the construction of correct programs, while "of course" that is what they should do. Instead opponents of testing advocate the use of formal verification, or even better programming in such a manner that faults cannot be introduced in the first place.

Much has been written about this methodological issue and several authors have defended intermediate positions. Nevertheless it seems to be the case that this kind of accusation has promoted voluntary non-cooperation of many theorists with the very large group of practitioners who held (and still hold) that testing is a necessity. An intriguing consequence of that situation that theorists engage in negative qualifications of the practical value of testing without providing (or making reference to) a proper definition of testing to begin with (for instance see [21]).

Conversely in the software engineering community opponents of formal verification have voiced the accusation that it is too expensive, if it can be done at all, so striving for formal specification and formal verification of programs would be a waste of precious energy and time. In a specific situation the accusation can be like: A (manager) accuses B (programmer trying to make use of formal methods) of "you are wasting our time and money". In this form the accusation can be quite influential, whether or not the body (you are wasting our time and money) is valid. On the basis of such fairly general, and sometimes more specific, accusations many programmers have been led to voluntary non-cooperation with proponents of the formal methods community often without having much awareness of what can and what cannot be done with such methods.

Rather new is the following accusation: A (academic manager or industrial research manager) accuses B ("old style" researcher in computer programming) of "working on outdated topics by not being visible in quantum computing". This situation is not at all hypothetical, several instances of this phenomenon are know to the authors.

3.3 A difference in style between PT and AT

Promise theory was conceived by Mark Burgess with the idea in mind that artificial agents (such as machines and programs) might just as well issue promises as human agents may do, but are less plausibly subject to obligations. This origin has led to promise theory having a bias towards promises made by and about artificial agents, as e.g. in [6]. For accusation theory it is, for the time being at least, more plausible to view accusations as involving human accusers and accusees, as well as agents in scope. We expect that this asymmetry is a temporary matter, and that at some stage artificial agents will be involved in issuing accusations. Perhaps that already takes place in some cases with automated production and distribution of messages on social media.

3.4 Accusations supposed not to create obligations for validation revisited

Suppose A accuses (with accusation p) B of β with scope S. And suppose that this situation is dealt with in court.

(i) If A is the prosecutor and B is the defendant then it is expected that A makes a best effort to show the validity of β. A is under no obligation to succeed for the simple reason that the court is to some extent free in its assessment as to whether or not A has succeeded in validating β.

(ii) If the fact that p has been issued by A is the case in court, so that, say C accuses A of having issued accusation p against B (and by doing so has committed a punishable wrongdoing), then it may very well be the case that:

(a) the court agrees with C in a case where β is valid (while the publicity given to that fact by A was wrong), or

(b) that the court disagrees with C while it holds β invalid because the court is unimpressed by the claim that A should not make an accusation of this kind, or

(c) that the court agrees with C while it holds β invalid because the court agrees that A should not produce an accusation of this kind without providing convincing arguments for β at the same time,

(d) that the court disagrees with C because after all it turns out that β is valid, while at the moment of producing accusation p that fact could not have been known to A (so that A apparently made the accusation by way of a gamble, not knowing whether or not validation of β would be required in the future an to what extent such validation would be possible).

In each of the above cases (a),..,(d) the postulate that an obligation to validate β has been created by A's issuing of p is not a decisive factor. A being obliged to "have the intention to demonstrate the validity of β" seems to play no role either as intentions are hard to assess for a court.

3.5 Promises about accusations

In the following situation A, B, C, D are supposed to be participants of an organisation, D has a management responsibility, A has procedural responsibility, while C and B are colleagues who are supposed to cooperate smoothly. The organisation maintains a classification of issues and γ is one of the categories in this classification.

The following pattern involves person C who accuses person B within an organization managed by D. In the organisation person A has a procedural responsibility for dealing with problems among participants.

Here is an example of a situation involving two accusations a_1, an accusation issued by C towards B, and a_2 a repeat of essentially the same accusation though now within a formal context where agent A has become responsible for handling the accusation as a problem within the organisation.

Promises are used for communication about and announcements of accusations. In this example promise p_1 serves as a warning by C that an accusation will be made, and with p_2 the B points out to C that they expect management (that is D) to become involved. Promise p_3 is an announcement to C that A will deal with the accusation in a systematic manner, most likely in accordance with internal guidelines of the organization. Promise p_4 contains the message to B that A will deal with the problem (of the promise having been made). Finally with a_2, C repeats their accusation against B, now within the context of the of the formal procedure as set up by A.

1. p_1: C promises B that "C will accuse B of an issue of category γ",
2. p_2: B promises C that "B will require that the issue, whatever it is, is first discussed with D",
3. a_1: C accuses B with scope $\{A, C\}$ of "an issue Y (with B) of category γ",
4. p_3: A promises C with scope $\{A, C, D\}$ that

   (i) A will refer to the accusation as case n, and

   (ii) upon having received more information about Y, A will deal with the matter in detail, and

   (iii) A will communicate the existence of case-n to B,

5. p_4: A promises B with scope $\{A, B, C, D\}$ the following:

   (i) C has accused B with scope S' of Y where Y is of category γ and is yet to be explained (to A) in more detail, and

   (ii) A will take the initiative to deal with the matter in detail upon having received said explanation, and

   (iii) A expects B to await further steps in this matter by A, and

   (iv) A will refer to the matter as case-n, involving accusation a_n, with body $Y = \beta(a_n) = \beta_n$".

6. a_2: C accuses B with scope $\{B, C\}$ of:

   (i) an issue Y of category γ (repeating accusation a_1, while making reference to case n), and

   (ii) the explanation that in particular Y amounts to β_n,

An instance of this case arises if C is a designer of some type of artifact who is in need of a program written by B and C accuses B of not taking the time tor turn B's requirements into a technical specification (with sloppy work as a consequence). This brief exchange of promises and accusations indicates various elementary facts:

- case descriptions in which accusations occur are likely to involve promises as well, and in fact the issuing of even a single significant accusation may be embedded in a plurality of related accusations,
- it is not uncommon for an accusee not to be in the scope of an accusation, (in such cases the accusation is called indirect). In the example B is not in the scope of accusation a_1.
- accusations my be anonymous as well (assuming that all agents involved are able to communicate in an anonymous manner) for instance:

  a_1^{anon}: C anonymously accuses B that "someone experienced (with B) an issue Y of category γ", (this anonymous accusation might occur as an alternative to a_1 in the mentioned pattern),

  A non-anonymous accusation is alternatively referred to as a signed accusation.

Anonymity and indirectness are independent properties of an accusation. We assume that an accusation is by default non-anonymous and is by default direct, so that anonymity and indirectness ought to be mentioned when informative,

3.6 Intentional ambiguity of the term accusation

The concept/term accusation features intentional ambiguity. One may understand of accusation as (i) an event of accusing that took place in space and time), or (ii) as one of a plurality of possible abstractions of said event, or (iii) as an abstract entity' which may not be traced back to a specific event of coming into existence.

Labeling this ambiguity as intended implies that if more resolution is needed such must be made explicit. For an intentionally ambiguous notion one is not looking for a more specific default meaning, under the assumption that in many cases the ambiguity can be resolved by taking the context of occurrence into account. For instance "p as issued on (place, time, circumstances)". Purely abstract use is for example as follows: "p will serve as a description of the accusation at hand, the history of p is not precisely known unfortunately but what we know is this: ....". Or if the committee is confronted with an accusation p of category γ the committee will proceed in three steps as follows...".

With concept/term we denote the idea of a concept being understood as referred to by a term together with the term being used for naming the concept. In many cases there is still a question about the meaning of the concept/term.

Our reason for pairing concept and term is to emphasise that when, for instance, asking for the meaning of animal/bat (and not baseball accessory/bat) one need not first go into excessive detail about the meaning of animal, but one can make use of the fact that the focus will be on animals which are plausibly labelled as bats. In accusation theory we have coupled the informal concept of an accusation with the term accusation. Now of course the informal idea of an accusation can exist without the term "accusation" (or a synonym of it) having been incorporated in the language. We have been looking for an equilibrium given the existence of the term accusation as well as its various connotations. We are not claiming that there is a unique concept of accusation which, as if by accident, we denote with accusation.

Another example is the term (in fact phrase) "natural numbers". When asked to explain "natural numbers" it is helpful to begin with stating that term will be understood as somehow denoting a class of numbers understood as entities in logic and/or in mathematics. Having arrived at that point one may on the one hand start looking for definitions which logicians and mathematicians have been giving to the term "natural numbers", and on the other hand looking for concepts for which "natural numbers" might be the most appropriate name. Finally one may or may not wish to choose one's favoured interpretation, or one may prefer to leave some (perhaps unambiguously specified) intentional ambiguity in place.

4 Options for accusables in the context of programs

We assume that A (programmer, seller of program X) and B (user, buyer of X) are agents who represent their respective organisations and who may have other agents and teams working for them. The program X may in fact consist of a plurality of programs. We will use the listing of accusables in 2.3 above as the primary organisation mechanism.

Regarding personal behaviour there is the familiar complaint that A would not be able or willing to discuss with B the problems which B is worried about:

> B accuses A of being unable or unwilling to discuss in non-technical language one or more problems which B has reported.

Accusations concerning a problematic professional action and inaction can be combined under the assumption that throughout these accusations inaction is an instance of problematic action (that is acting too late). Now a classification with respect to the phase in the software lifecycle can be used. First we list accusations in both direction in connection with requirements and specifications.

Contracting phase:

B may accuse A of: *overestimating its own capabilities and uniqueness*:

- having made the impression of having ample experience with projects of this type (while that was not at all the case),
- having insufficiently qualified staff for the project to implement S,
- having too few qualified staff members for the project to implement S, (other problems: illness, overworked etc.),
- asking too much money for the development of X,
- charging excessive cost/hr during a significant part of the planned project,

A may accuse B of: *underestimating the challenge involved*:

- of offering not enough money to perform job (developing, validating, and implementing S_{spec}),
- setting an unreasonable deadline (too much time pressure),
- not being willing to pay for the best staff which A has on offer,
- expecting from A an excessive degree of outsourcing the work to low paid workforce in other organisations.

Requirements capture and specification phase:

1. A accuses B of having provided a *problematic requirements description* R_{req}, more specifically A accuses B of:
   - not providing informative replies to their questions $q_1, \ldots, q_n$,
   - not having used a systematic and known process for requirements capture so that A finds it hard to grasp the completeness of the result,
   - not having grasped the importance (and cost) of prototyping for parts of the requirements capture process,
   - having changed the interpretation of certain fragments in R_{req} with respect to the interpretation which was assumed in previous stages of the process,
   - implicitly demanding that A produces advice concerning requirements capture without making that demand explicit and without providing adequate compensation for it,
   - Not allowing A (by way of limited project funding) to achieve a formal specification of the behaviour of the program to be produced,
   - Requiring that A will design X as the implementation of an algorithm P (that supposedly is compliant with R_{req}) and which B has transferred to A in the form of a flock of algorhymes (see [8] for this notion); (now A may accuse B of) having provided a faulty algorithm (see [8]).
2. In connection with formal verification: B accuses A of one or more of the following *dysfunctional uses/misuses/non-uses of formal methods*:
   - having missed a relevant and cost-effective opportunity for formal specification/verification,
   - having used (for the purpose of verification) a formal specification which does not match with the intended specification S_{spec} (deviation being relevant for B, however),
   - having written informal proofs of significant properties of the specification without making the effort to formalise the proof and to have the proof machine checked (which is the state of the art in such cases),
   - having spent too much time and money on (failed) attempts to apply formal methods,
3. B accuses A of having provided a *flawed specification* (that is a specification involving any problem that may potentially or will necessarily result in failing implementations) as a proposed solution of the requirements R_{req} as provided by B, more specifically:
   - having provided an unrealisable specification (in which case the specification is inconsistent, or stated otherwise the specification has (suffers from) a consistency flaw),
   - having provided a specification which is too weak and which for that reason admits implementations which do not comply with the given requirements (in which case the specification has an underspecification flaw),

- having provided a specification which is plainly wrong by admitting failing implementations only, with respect to R_{req}, in which case the specification features a misspecification flaw,
- having provided a flawed specification which in fact contains one or more faults (known to B), that is a limited number of fragments with proposed local changes thereof in the (text of) S_{spec}, the application of which can bring about an adequate or less inadequate specification S'_{spec}, so that these fragments may be considered to cause a flaw.

Accusations in connection with product quality:

1. B accuses A of having provided B with a program X with a *quality problem*. More specifically B accuses A of one or more of the following deficiencies:
   - having shipped X which was flawed at the time of delivery (that is non-compliance with requirements; counterexample known to B),
   - of having provided a failing program X. (Here failure is non-compliance with specification S_{req}; a witness/primary test case of which X fails is known to B),
   - having provided a buggy program X; in return A may accuse B of
     - complaining about buggyness without being specific about the nature of these bugs: are these failures or faults.
     - complaining about buggyness (in particular the occurrence of failures) without providing proper test cases for documentation of these,
   - B accuses A of having produced a faulty program X (that is a program that contains one or more faults (known to A); a witness/primary test case is known to B with evidence that A knew about it); in return A may accuse B of *gratuitous complaints about buggyness*, for example:
     - accusing A of delivering a faulty program while in fact the statistics that have resulted from the testing phase, as well as the statistics which are obtained regarding the initial phase of use are quite good given the type of program X,
     - having accused A of delivering a faulty program X without being aware of a scientifically well-understood definition of a program fault, and of the occurrence (in a program) of a plurality of program faults,
   - having produced a program (X) which is: lacking structure, lacking modularisation, lacking proper documentation, and which for that reason is poorly accessible to maintenance; in return A may accuse B of:
     - not having mentioned these issues in the original requirements specification (plausibly to be listed under the heading nonfunctional); in return B may accuse A of: one or more of the following:
       - evading their own professional responsibilities,
       - not asking B for consent on critical issues where such interaction would have been expected (by B) as a feature of the normal business process,
       - not having improved on matters of style and appearance in spite of this issue having popped up in a recent past with work done by A for another client of A (who happens to be known to B),
     - having outdated notions of structure and maintenance in mind,
     - not having given any prior feedback on these matters during the development process although there would have been ample opportunity for doing so,

2. B accuses A of one or more of the following *problems with style and appearance* of X:
   - unfortunate choice made of the program notation used for X,
   - unfortunate choice made for a development environment for the project to implement S (that is leading to X), (in particular some plausible tools were unavailable, or not available at the needed level of quality or maturity, so that the development technology has not been state of the art),
   - having carried out a program (X) (development) process which is: lacking historical documentation and data, had insufficient software support, lacking proper documentation, poorly accessible to maintenance,
   - not having chosen a well-defined software process model (so that problems with R might have unnecessarily gone unnoticed),
3. Intellectual Property Rights (IPR) related accusations regarding the quality of X in terms of *IPR and ethics (responsible programming)*:
   - C accuses A of having used (copyright infringement, including misuse of open source code) an IPR protected program (with IPR/copyright belonging to C) in the construction of X,
   - B accuses A of it having become vulnerable to third party IPR based claims because of copyright and/or patent infringement(s) in connection to X which become vulnerabilities relevant for B upon adoption and use of X by B,
   - C accuses A of having used (patent infringement) an IPR protected algorhyme (see [8]) (with IPR/software patent/algorithm patent belonging to C) in the construction of X,
   - B accuses A of having delivered X including a moral defect (see [8] for that notion).

Accusations in connection with process quality:

1. B accuses A of *poor technology choice*, in particular one or more of the following :
   - making a mistaken choice for the program notation for X,
   - making a mistaken choice for a development environment for the project to implement S (that is leading to X), (in particular some plausible tools were unavailable, or not available at the needed level of quality or maturity, so that the development technology has not been state of the art),
   - having performed program (X) (development) process which is: lacking historical documentation and data, had insufficient software support, lacking proper documentation, poorly accessible to maintenance,
   - B accuses A of not having taken IPR matters seriously, with the effect that the legal status of X is may be at risk,
   - B accuses A of not having sought IPR protection (in an appropriate form) for X as delivered to B while knowing that B aspires that such protection will be available,
2. B accuses A of negligence of *explicit software process modelling and compliance policing*:
   - not having chosen a well-defined software process model (so that problems with R might have unnecessarily gone unnoticed),
   - having chosen a definite software process model, say $\text{SPM}_{R_{req}}$ but having failed to enforce compliance of the actual development process for X with $\text{SPM}_{R_{req}}$ so that a software process flaw (see [10] for the notion of a software process flaw) has occurred during the development of program X. More specifically B accuses A of:

- B accuses A that their software process management failed to prevent (deliberately?) the occurrence of a software process flaw during the development of X (resulting in a flawed but not faulty program X).

3. B accuses A of *inadequate testing and verification*, in particular one or more of the following "accusables":
   - having conducted inadequate pre-release testing (verification) of program X, more specifically:
     - not having taken the time to develop an expected user profile (on which to base black box testing),
     - not having tested each instruction (no full line coverage) and not having tested each branch (no full branch coverage),
     - not having taken the time to perform fuzzing in search of security vulnerabilities,
     - having missed the opportunity to perform metamorphic testing on a significant scale while the specification suggests ample opportunity for such forms of testing,
     - not having considered the possibilities for performing Risk Based Testing (RBT),
     - having conducted poor performance testing,
   - not having made an estimate of the number of dormant failures in X at the time of first release, (see [7] for the notion of a dormant failure),
   - not having made an estimate of the number of dormant faults at the moment of first release, In return A may call into question the very notion of a dormant fault: with this accusation:
     - A accuses B of "accusing A of shipping X with a significant number of dormant faults" while B is not even aware of a convincing definition of the notion of a dormant fault,
     - A accuses B of making up ad hoc acceptance criteria which ought to have been covered and documented in the requirements capturing phase,
   - not having performed formal verification in cases where that would have been plausible and possible,
   - not providing insight in protocols and data regarding pre-release testing of program X,
   - having disregarded warnings by their own staff members regarding potential (post-release) failures of X,

4. B accuses A of *poor post delivery service*:
   - not responding to a specified collection of B's bug reports,
   - responding too late to a specified collection of B's bug reports,
   - not having improved X in order to deal with a specified collection of B's bug reports,
   - having released program X while a number of last minute bug reports had not yet been handled and fixed; and not being clear about that state of affairs when releasing bug fixes seemingly unrelated to B's bug reports emerging from actual usage,

5. A accuses B of having *underperformed as a commissioning party* (that is client for a custom made program), in particular A accuses B of one or more of the following accusables:
   - having provided unrealisable requirements R_{req},
   - having failed to provide useful answers when clarification of R_{req} was needed,
   - having failed to keep in touch with the development process, while being aware that they would be insisting on choices which were not made explicit beforehand,

- expecting that A takes IPR risks in order to reduce the cost of the project,
- having asked A to develop an implementation of R_{req} wile doing so unavoidably leads to a program with a moral flaw (see [8]),

Security related:

- Security related accusations: B accuses A of:
  - having (consciously?) delivered a program (X) which when installed on B's platform creates a significant (and unnecessary) security risk,
  - not having taken standard precautions it safeguard B (when using X) against security problems for instance:
    - not having provided in time a security patch after a (new and thus far unknown) security flaw in underlying system software has been announced world wide, (example: Log4J2)
    - by not facilitating the installation of an upgrade/patch of underlying systems as soon as these are distributed by the respective engineering teams,
    - by including open source software in such a manner that following the maintenance path of that software is impeded,
  - not having responded in time to B's question whether or not a recently discovered security flaws for system software entails a risk for users of X,
- B accuses A of having disclosed (security relevant) confidential information about its activities (for example contents of R_{req}, or subsequent communication between A and B) to some other agent C.

4.1 Assuming a hypothetical program statistics background

We may assume that in recent years say starting y_{-k} until last year y_0 a number of a_k programs were delivered, with a length distribution as follows $l(k, m_1.m_2)\%$ of the a_k programs (say Y) delivered in year k have $m_1 \leq LLOC(Y) \leq m_2$. We assume that a program is produced with a certain quality level in $\{low, moderate, medium, high, critical\}$. We do not distinguish between the statistics of different manufactures. Now assume that quality level high has to be achieved and let $n_{high}(k)$ be the number of such programs delivered by A with intended quality level high.

We assume that independent of the LLOC of the latter programs the number of failures and underlying faults spotted and repaired per instruction (in the first year of its deployment) were fairly constant, say 2 per $10,000$ instructions. Suppose that $LLOC(X) = 250,000$ then some 50 failures with corresponding faults were found and changed in the first year of use of X. It then follows that the accusation 1 (bullet 4) of "accusations in connection with product quality" above is irrelevant without quantitative information about the number of faults.

Proposition 4.1. *An accusation to the program manufacturer A about the existence of faults in program X (using the name as an identifier which points to the most recent version of X as delivered by A) MUST involve quantitative information on the number of faults and must refer to plausible bounds for such numbers which supposedly have been exceeded.*

It follows from the above Proposition that the notion of a Laski fault is unlikely to be useful (for issuing accusations about program quality) and that one must think in terms of numbers of MFJ faults (see [20] a first paper proposing a precise definition of the number of faults in a program), or in terms of numbers of RTJoC faults (which is not a defined notion in [4] where RTJoC faults were introduced, though a similar definition as for MFJ-faults makes sense).

5 On the relevance of accusation talk for programs

Suppose one holds that program X must be delivered by A in such a manner that compliance with S_{req} is formally demonstrated and the formal proof has been checked by computer. Now there is no room for accusations about failures of X and faults in X, or about the way in which testing has been performed. Some conceivable accusations which were mentioned above may indeed disappear once better production technologies are adopted. Not all accusations can be expected to disappear in that manner. Nevertheless we will look for arguments which motivate the use of accusation talk.

Let Y be an artefact in class arteFact (example of artefact class, just as X is an artefact in class program). We assume that there are various characteristics χ on artefacts of class arteFact and we assume that $degree(\chi, Y) \in \{good, ok, marginal, notok, bad\}$ measures the degree to which an artefact Y complies with given expectations of its compliance with characteristic χ. The engineering challenge comes about to design and construct Y so that it meets certain requirements R and say $degree(\chi, Y) = ok$. Computer programming (taking program for arteFact) abounds with such challenges, with $\chi \equiv \chi_{corrrect}$ stating of a program that it is correct as a prime example. In practice one often considers $\chi \equiv \chi_{reasonably-fault-free}$ and then computer science has deplorably little to say about the technical meaning of this characteristic.

Nevertheless much can be said without thinking in terms of accusations. Given Y a user of Y may come to think that in fact $degree(\chi, Y) = notok$ for some important characteristic χ of arteFact's. The assertion "$degree(\chi, Y) = notok$" may be understood as an anonymous complaint about Y without addressee. It follows that by contemplating such assertions, understood as negative assessments, some notion of a complaint comes about without any need to introduce agents by whom or to whom a complaint is issued. Most of software engineering theory is configured in that manner: no mention is made of the various agents who are complaining and who are arriving at judgements of the form $degree(\chi, Y) = d$ for some $d \in \{good, ok, marginal, notok, bad\}$. Our task is twofold: (i) arguing that it is useful to think of a complaint as being issued by an agent and being received by another agent, perhaps with some agents in scope, and (ii) to argue that in addition to complaints there is ample room for accusations, and (iii) to argue that in the context of programming accusations have a role to play.

5.1 Complaints and bare complaints

Definition 5.1. *A bare complaint (with respect to an artefact Y of class arteFact) is an assertion of the form "$degree(\chi, Y) = d$" with $d \neq good$.*

Definition 5.2. *A complaint (with respect to an artefact Y of class arteFact) is a bare complaint β (the body of the complaint) together with:*

(i) an agent (say C) who is complaining (complainant, complaining party),

(ii) an agent (say E) who is a party to whom the complaint is directed (target of the complaint),

(iii) a collection S of agents in scope (who are made aware of the complaint).

It may be the case that one or more accusations are contained in the body of a complaint. For instance:

> A complains to U (an external body) that "A accuses B_1 of β_1 (A, U in scope)" AND "A accuses B_2 of β_2 (A, U in scope)" AND $degree(\chi, Y) = notok$", with A, U in scope of the complaint.

The advantage of thinking in terms of complaints is immediate in circumstances where judgements of the form $degree(\chi, Y) = notok$ are highly subjective. By thinking in terms of a complaint several agents are introduced the judgement of whom will be taken into account. In computer programming the assertion

"program X is reasonably free of faults" is highly subjective. Such judgements can only be understood in the context of a specific tradition of the production and use of certain classes of programs. Thus while the science of computer programming is written and conducted in terms of bare complaints and how these are to be avoided), in the practice of computer programming there is no escape from the introduction of agents who play a role in arriving at notoriously subjective assessments.

The list of possible accusations in the context of programming qualifies at the same time as a list of possible complaints. The principled conclusion which we draw from the list is that it is obvious that subjective judgement has a large role to play in programming, so that thinking in terms of complaints and not merely in terms of bare complaints is plausible and warranted.

5.2 From complaints to accusations

We consider once the situation that programmer A has produced program X and that user B comes to complain that X suffers from failure f, which is caused by fault F_f. Now it is obvious that only A is to be blamed for making the mistake of having an instance of fault F_t in X which causes that failure f can (and by definition of failure sometimes will) take place when running X (that is on some inputs, not necessarily on all inputs). We write $X \underline{features{-}failure}\ f$ for the assertion that X shows failure f. Computer science offers may ways to denote that fact.

Now consider the logic of agency as developed for example in [23] and [17] with the key modal operator $E_-\ E_A\ \phi$ denotes that A brings about that ϕ holds. If A is the programmer of X then it is reasonable to assume that $E_A\ (X\ \underline{features{-}failure}\ f)$. In particular if fault F_f causes $(X\ \underline{features{-}failure}\ f)$ it may be claimed that A made a mistake by writing program X in such a manner that (simply) repairing fault F_f (by replacing its footprint with the corresponding change, that is fixing the fault) would improve the behaviour (when running) of X. The literature on computer programming seems to be unanimous in attributing the origin of faults and consequential failures to the mode of operation of programmers. The underlying assumption is that: programmer mistakes lead to programs which contain faults that are causes of failures. Fixing a fault can be understood as a form of backtracking from the side of the programmer.

The user enters the picture in a different manner. More often than not specifications are somewhat vague and incomplete. However the specification is expected to be clear about "what matters". Now it is up to the user to determine "what matters", and in this manner the very determination of failures becomes linked to the user. Only failures which are (or might) ever taking place during use "really matter". We find the both programmer and user as well as their respective judgements enter the picture. Programmers must find out by themselves what it is that matters in a specification. Programmers also must entertain a view on what it means that $(X\ \underline{features{-}failure}\ f)$, and importantly, which forms of failures f must be taken into account.

5.3 Acceptance testing versus formal verification

Remarkably in a context where the assertion that X is a correct implementation of S_{spec} can be formally shown and where it can be taken for granted somehow that an implementation of S_{spec} will also be an adequate implementation of R_{req} the promise "A promises B that the development of program X has been finished and that X will be usable (by B) as an implementation of requirements R_{req}" can be decomposed as: (i) A has developed X and has formally proven that X implements specification S_{spec} AND (ii) B holds that implementations of S_{spec} will meet the requirements R_{req}.

It turns out that, at least in principle, the use of formal methods allows a complete separation of concerns between programmer A and user B so that no primitive relation involving both agents is needed, in particular

not a relation that describes either promises, or complaints, or accusations, each of which involve two agents.

However, in practice it is very often the case that a programmer knows for whom (that is for what kind of use) a program is written and in such cases acceptance testing is done with an expected user profile in mind, moreover B's taking a new program on board takes place on the basis of (i) the confidence that R_{req} adequately captures what B needs, (ii) the confidence that A has acquired (primarily via testing) that X implements S_{spec} as well as R_{req}, (ii) the trust that B has in developer A, and (iii) the expectation of both A and B that bugs found by B will be adequately communicated to A speedily resolved by A.

5.4 The case for contemplating accusations on programs

We believe that the relevance of the promise relation in the context of computer programming, say A promises B that program X will serve B's plans, has been argued for convincingly.

Now computer programming is to a high degree a matter of trial and error, and therefore availing of flexible ways of dealing with adverse circumstances. Thinking in terms of accusations provides a flexible language for such matters.

6 Third party judgement of accusations

Whenever an accusation p with body β is made agents in scope may wish to assess the situation. Various discussions in such a context directly discuss whether or not the accusations are false. Accusation Theory in the sense of [11] starts from the assumption that it is necessary to first reflect on the status of an accusation. In that sense one first has to ask whether or not it is justified to make an accusation at all before one can investigate whether or not an accusation is valid or not. To say 'an accusation is justified' says nothing about the falsity or validity of an accusation but only states that it is in some sense reasonable that an accusation is made. If that is not the case, the question of validity did not even occur. In that sense we will distinguish the following judgements about an accusation p:

- p is justified: it is in some sense reasonable that the accusation is made (a precise description of justified accusations is given in Definition 6.2 below.
- p is not justified: it is not considered reasonable that the accusation is made,
- p is valid: β is validated, that is the content of the accusation holds true,
- p is invalid (that is not valid): β is refuted: that is the accusation is (considered to be) false,
- p is evidence immune: there is no systematic method to determine the validity or invalidity of p.

6.1 Evidence immune accusations

An accusation p with body β is unprovable if the validity of β cannot be established by any reasonable methods of proof. An accusation is irrefutable if the validity of β can not be refuted by any reasonable methods of proof.

The situation can be more complex: an accusation may be unprovable because either the proof that is found is too weak, or because a method of proof that often works "in such cases" does not apply in this particular case. Similarly with the accusation being irrefutable.

More importantly an accusation can be both categorically unprovable and categorically irrefutable: in such case it is very implausible that a proof can be found in either direction.

Definition 6.1. *An accusation p is evidence immune if β is both categorically unprovable and categorically irrefutable.*

We have not found thus far a plausible case of an evidence immune accusation in the context of computer programming.

6.2 Justification of accusations

Assume that (with accusation p) A accuses B of β with scope S. Now let $C_{handling} \in S$ be an agent who is supposed to be formally handling the accusation. It is plausible that the accuser has definite expectations on how the accusation will be received by their friends in scope, as well as by their non-friends. It may at the same time be unclear to the accuser how $C_{handling}$ will deal with the matter.

When is an accusation justified? In promise theory it is assumed that whether or not a promise is kept is a matter of assessment by individual agents, and that there is no universally valid notion of a promise having been kept. Similarly

Definition 6.2. *(Justified accusation) Assume that (with accusation p) A accuses B of β with scope S and $C \in S$. Now accusation p is justified according to C if (i) C considers β to express content that merits being the subject of an accusation from A to B, and moreover (ii) one of the following criteria is met:*

(a) either C considers β to be factually adequate, or otherwise

(b1) C considers it plausible to such an extent that β is factually adequate that C considers the accusation to constitute a reasonable instrument (from the perspective of A) for obtaining clarity on precisely that matter, or otherwise:

(b2) C considers it plausible that (although A doubts that β is valid, or even if A knows that it is not) from the perspective of A it is plausible to such an extent that β is factually adequate that C considers the accusation p to constitute a reasonable instrument (from the perspective of A) for obtaining clarity (for A) on precisely that matter, or otherwise:

(c) C considers the virtue of the accusation in its capacity of a warning sign (to agents in S, but conceiving of β merely as a relevant pattern not necessarily related to A or to B, see "warning signal rationale") sufficiently high in order to hold that the accusation can be maintained even when factual adequacy of β is likely to be unwarranted or is unlikely to be properly investigated, or otherwise

(d) C knows that β is both categorically unprovable and categorically irrefutable, but C has so much more trust in A than in B that C considers it justified that A accuses B by way of an evidence immune accusation.

(e) A contemplates a neutral complaint about B concerning β. However, by issuing an accusation A consciously risks some increased vulnerability (A expects to be held to account), more so then will be the case with a complaint (so A expects); in such circumstances A deems an accusation to be less aggressive than a corresponding complaint would be.

With case (e) it is not meant that in general issuing an accusation would be less aggressive than making a corresponding complaint, only that in some cases that might be the case which then might be considered to constitute a justification of the accusation at hand.

6.3 Validation of accusations

Validation of an accustation with body β is a matter of validation of β. In many cases justification of an accusation precedes validation. In court an accusation is justified by the very fact that it is put forward

in court. The proceedings in court have the objective to arrive at either validation or invalidation of the accusations that have been made.

In computer programming user B may be unhappy with a program X written by programmer A and then (accusation p) accuse A of having written a buggy program. Such an accusation is justified by the very fact that B is unhappy about the functionality of X when put in use. However, a subsequent attempt to validate accusation p may lead to the conclusion that B has provided A with a requirements specification which, in hindsight, has proven defective, it fails to capture the intentions of B.

6.4 Controversial instances of justification of an accusation

Of these grounds for accepting a justification for accusation p the criterion $(ii) - c$ may be controversial, and the same holds for criterion (d).

6.4.1 Doubts on justification in terms of the general interest.

If C communicates a positive judgement of an accusation (which can be done for example by means of a promise to the accuser or to the accusee with the other agents in scope), and if justification $(ii) - c$ is applied by C then B may be inclined to accuse C of misusing the accusation issued by A.

This situation has many different forms:

(i) C may not mention factuality of β and disregard the need (for instance by being satisfied an unconvincing judgement by $C_{handling}$ of the matter) for significant further investigation to such an extent that B is bound to assume that ground $(ii) - c$ has played a major role for C,

(ii) C may promise to be convinced of the justification of p on ground $(b1)$ or $(b2)$ while in fact being motivated to express their support for A on the basis of a consideration of the form $(ii) - c$.

6.4.2 Doubts on justification on the basis of loyalty.

Justification of an accusation along ground (d) may also be controversial but in fact there is a group of philosophers (who are communicating via the PEASOUP newsletter) who advocate precisely this mechanism. They suggest that justification ground (d) may be admissible under the constraint that those who accept the accusation by judging it justified must engage only in limited actions (perhaps retaliation) against B. B may only be deprived of "goodies" to which B is not entitled.

In the setting of programs one may imagine that the complaint is about a program X that B has bought from A being buggy (according to A). Now C is the scope of the accusation may choose not to acquire an instance of X from B. Recommander systems (public reviews etc.) support this mechanism.

Justification by C of p on the basis of (d) comes (for C) with an awareness of victimhood of B. Victimhood may or may not originate spontaneously from an accusation.

Definition 6.3. *(Trivial victimhood.) Assume that (with accusation p) A accuses B of β with S in scope. Now with this accusation comes (by definition) with a promise of victimhood: A promises to be victim of B on the basis of p.*

Definition 6.4. *(Reciprocal victimhood.) Assume that (with promise p) A accuses B of β with S in scope. Now with this accusation comes (by definition) a possibility for a promise of victimhood for B: A promises to be victim of A on the basis of the claim that p is an unwarranted accusation.*

CLAIM: the promise of reciprocal victimhood does not automatically come into existence (unlike the promise of trivial victimhood). The need for B to issue a promise of reciprocal victimhood is often underestimated. Only if the promise of reciprocal victimhood is made by B is it plausible that B would complain about C judging positively about p on the basis of grounds as in $(ii)-c$ of Definition 6.2.

6.5 A case involving two accusations and two promises

Below we discuss an accusation pattern where an accusation leads to a promise issued by an agent in scope, then a second, similar, accusation leads to a similar (though less detailed) promise by the same agent in scope. Now the accusee of the second accusation may be unhappy, when assuming that the grounds for making the second promise where roughly the same as the grounds issuing the first promise.

Assume conditions (1),..,(6):

(1) (with accusation a_0) A_0 accuses B_0 of β_0 with C in scope S_0,

(2) assume further that (with promise p_0) C promises A_0 of having observed and concluded that:

- A_0 made accusation a_0,
- (according to C) the grounds as mentioned in $(ii)-c$ (of Definition 6.2) suffice for a justification of accusation p_0 in view of the number of persons who suffer similar victimhood as expressed by β_0,
- that the above consideration (involving $(ii)-c$ of 6.2) matters irrespective of the actual state of affairs in the case of accusation p_0,

(3) moreover, (with accusation a_1) A_1 accuses B_1 of β_1 with C in scope S_1,

(4) β_0 and β_1 are similar in terms of what the accusation is about,

(5) and (with promise p_1) C promises A_1 of having made accusation β_1 with scope S,

(6) now using abduction B_1 concludes that C had in mind grounds $(ii)-c$ of 6.2 when issuing promise p_1 (while B_1 strongly disagrees with these grounds).

Under these assumptions it is plausible that B_1 is dissatisfied about C and contemplates further moves, such as a formal accusation in court based on the viewpoint that the reputation of B_1 has been damaged by C.

6.5.1 Refinement of the case to a case about programs

This case has an interpretation in terms of computer programs as follows, where we assume that $B = B_0 = B_1$. B has produced and sold a program X to a large group of users. With accusation a_0, A_0 accuses B of having shipped X while still containing too many failures in part F_0 of the functionality of F. C is a user of X who is quite dissatisfied with X, so much so that C is happy about the accusation made by A_0 and she says so, by means of a promise which suggests that many users are dissatisfied with X while paying no attention to the validity of the particular accusable β_0 as provided by A_0 in p_0. Then with accusation p_1, a second user A_1 of X, accuses B of having shipped X while still containing too many faults in a part F_1 of the functionality of X. Again C pronounces support for the accuser, this time A_1, by way of an accusation, now without any further comment.

Now B may disagree with A_1 and may be quite certain that the functionality F_2 is not defective (no failures observed), and B may hold against C that C has procused accusation p_1 while not having confirmed the validity of $\beta(p_1)$ so that the reputation of B has been damaged by A_1.

6.5.2 Refinement of the case to a case about alleged problematic behaviour

The case has also an interpretation with $B_0 \neq B_1$. Now assume that $\beta(p_0)$ is a complaint about a certain type of personal misbehaviour, and that $\beta(p_1)$ is quite similar, though the persons involved are different.

C has issued a promise in support of A_0 with grounds $(ii) - c$ in view of the fact that such misbehaviour occurs frequently and that accusation p_0 serves the public interest by raising awareness of such problems. Accusation p_1 comes later and involves different people. Now for B_1 the promise q_1 is very unpleasant and B_1 guesses (an act of abduction) that C has based their verdict of the matter (as procused) on the same reasoning as in q_0 thereby exposing B_1 to reputational damage without paying much attention to the validity of $\beta(p_1)$.

6.6 Validation of an accusation

Validation of an accusation is a matter of subjective assessment by individual agents just as much as justification of an accusation.

Definition 6.5. *(Valid accusation) Assume that (with accusation p) A accuses B of β with scope S and $C \in S$. Now accusation p is valid according to C if (i) C considers accusation p to be justified, and moreover (ii) one of the following criteria is met:*

(a) C considers β to be factually adequate on the basis of their own information, or otherwise

(b) C considers β to be factually adequate on the basis of information available to a trusted third party, as well as the opinion about that as promised (with C in scope) by said third party.

Validation and justification for accusations are not identical and are not disjoint. Justification is about the "accusation as such", while validation is about its content. These notions coincide in the eyes of an agent who is not interested in the factual quality of β (which is quite often the case). This terminology leaves room for much confusion, which is intentional. in practice accusations do often lead to confusion and in order to make sense of such forms of confusion it is unhelpful to design a terminology which would by itself prevent that confusion arises.

This idea is comparable to the notion of program correctness. The very notion of program correctness makes sense only in a world where incorrect programs can be constructed. If means of construction for programs by definition assume the presence of a specification and guarantee by technical means compliance of an implementation with a specification then the very notion of program correctness becomes futile.

Definition 6.6. *Agent C considers a complaint containing an accusation p dismissed if C considers p unjustified.*

A complaint is rejected if it has not been dismissed while it has been rejected.

Definition 6.7. *Agent C considers an accusation p rejected if (i) C considers p justified, and (ii) C considers p to be invalid.*

Agent C may first come to believe that p is justified and only later develop the belief that p is not valid.

6.7 The buggy program accusation

The simplest and perhaps also most frequent accusation about a program is as follows:

Buggy checkout accusation: (Customer/user) B accuses (programmer) A (with scope U) of checking out (and handing over to B) a program X supposedly implementing S_{req} but unfortubnately quite buggy.

In symbolic notation this accusation look thus: $p \boxtimes_B^{A,U} [X \underline{\texttt{buggy-impl-of}}\ S_{req}]$. The question arises as to when the accusation may be considered justified, say by agent $E \in U$, and when it may be considered valid. We may discuss this as if the matter takes place in practice.

1. Computer science does not offer a definition of buggyness, and therefore an objective reading of the accusation is implausible. An epistemic reading is most plausible: according to what B knows about S_{req} in particular and program manufacturing in general it is the case that: X $\underline{\texttt{buggy-impl-of}}$ S_{req}. Other observers (for example agents in U) may arrive at different judgements.

   B may have heard from some $C \in U$ that A often delivers buggy programs and B may use "induction" to guess that X is buggy as well. E may consider this argument adequate for justification of p.

   In return A may accuse B of not making up their own mind on the quality of X and merely listening to gossip instead: $\boxtimes_A^{B,U}[B \texttt{ merely listens to gossip}]$.

2. B may have found during its use until the moment of issuing accusation p, a series of cases of input data, say $d_{in}^1, \ldots, d_{in}^k$ such that $\neg S_{req}(d_{in}^i, X(d_{in}^i))$. B considers k to be large given the limited time it took to collect these. C may consider the finding of k bugs a justification for the accusation.

   Now A may perhaps react with a counter accusation:
   $\boxtimes_A^{B,U}[B \texttt{ exaggerates: a single fault causes all listed failures}]$

3. B may have done more work and may have spotted k failures as in item 2 and may have identified as many MFJ-faults (including proposed changes, see [4] for MFJ-faults) each causing (and upon performing the change) solving one of these failures. C may see justification of accusation p one the basis of such concrete information.

   Now A may perhaps react with a counter accusation:
   $\boxtimes_A^{B,U}[B \texttt{ exaggerates: } k \texttt{ faults is ok given the number of instructions of } X]$. If this hypothesis can be confirmed accusation p is not valid.

4. B may have experienced failures of X so often that the use of X had to be put to an end, at least temporarily. C may agree that if X cannot be used in practice because of bugs it is buggy. B sticks to the accusation which is likely to lead to further investigation of X with the conceivable outcome that a small number of MFJ-faults can be held responsible for the failures that hamper use. In the latter case the accusation is not valid after all, although it was justified.

   A may react by accusing B of excessive extrapolation of the failures in X which B has found.

5. B may have experienced failures of X which were easy to handle. Nevertheless B has withdrawn X for the moment in order to investigate the quality of X, from which C may agree that if X cannot be used in practice because of bugs, it must be buggy.

   A may react by accusing B of not taking the standard maintenance cycle seriously. The normal workflow would have solved the problem that came out of these bugs quite speedily.

6. B may be right in warning other agents (in U) against the use of X, even if the B has not yet spotted many bugs. Nevertheless the bugs which B has found may be of such nature that issuing a warning (by way of accusation p) is justified to such an extent that the accusation is considered valid by C.

   A may accuse B of damaging the good name of A without convincing grounds.

7. Validation of p requires that B demonstrates that given the difficulty of S_{req} and given $\texttt{LLOC}(X)$, that is the logical lines of code (that is the number of instructions of X), having found k faults in (say m months) is relatively high. This demonstration must be based on a survey of literature with relevant statistics of programming.

6.8 Requirements specification versus technical specification

An important promise in practice is the technical specification checkout promise (as mentioned above):

> (promise) p: (programmer) A promises $customer/user$ B that S_{tech} constitutes an adequate design for an implementation of S_{req}.

The idea of A is that a program X which implements S_{tech} will also implement S_{req}. In other words:

$$\forall_{\texttt{program}} X.[X \ \underline{\texttt{sat}} \ S_{tech} \implies X \ \underline{\texttt{sat}} \ S_{req}]$$

If real time control of an embedded system is the functionality to which X is supposed to contribute then nomical necessity is the best A can hope for while metaphysical necessity is out of the question and is not intended to be achieved. What S promises involves:

$$\Box_{nomical} \forall_{\texttt{program}} X.[X \ \underline{\texttt{sat}} \ S_{tech} \implies X \ \underline{\texttt{sat}} \ S_{req}]$$

In practice A will use its own models and methods for system analysis in order to verify the nomical validity of the implication that a program which implements S_{tech} will also implement S_{req} so that epistemic necessity of nomical necessity will be claimed:

$$\Box_A \Box_{nomical} \forall_{\texttt{program}} X.[X \ \underline{\texttt{sat}} \ S_{tech} \implies X \ \underline{\texttt{sat}} \ S_{req}]$$

Here $\Box_P \phi$ expresses that according to what P knows ϕ must be the case. For chaining of such modalities we refer to [24] where these matters are studied in detail. Further A's claim is likely to depend on defeasible methods for asserting $X \ \underline{\texttt{sat}} \ S_{tech}$ (in spite of the fact that the latter assertion is in principle a purely mathematical matter). Let $\Box_A^{>t} \phi$ denote that A is sure that ϕ with a degree of certainty above $t \in [0,1]$ and let $\Diamond_B^{>r} \phi$ denote that B may convince themselves that ϕ is the case with certainty above r. Now A's technical specification checkout promise can be read as meaning that for some appropriate t and r (both known to A) it is the case that:

$$(\Box_A^{>t}[X \ \underline{\texttt{sat}} \ S_{tech}] \ \& \ \Box_A \Box_{nomical} \forall_{\texttt{program}} X.[X \ \underline{\texttt{sat}} \ S_{tech} \implies X \ \underline{\texttt{sat}} \ S_{re}]) \implies \Box_A \Diamond_B^{>r}[X \ \underline{\texttt{sat}} \ S_{req}]$$

When performing the technical design of real time embedded software the logical complexity of a seemingly simple specification checkout promise is remarkable. An optimal final state of the programming project is as follows: an inSq X has become available such that:

$$\Box_B \Box_{nomical}[X \ \underline{\texttt{sat}} \ S_{req}]$$

The optimal is unachievable and A and B must settle for, say:

$$\Box_B^{>s} \Box_{nomical}[X \ \underline{\texttt{sat}} \ S_{req}]$$

for some well-chosen $s \in [0,1]$. In these circumstances it is still possible that B finds input data d_{in} for X on which X fails to perform in conformance with S_{req}. If that happens B may accuse A of delivering inadequate work etc.

7 On the use of unjustified accusations

Assuming that whether or not an accusation made by A is justified primarily resides in the arguments that A maintains we find the following definition of an unjustified accusation (where the various negations have been inserted in capitals).

Definition 7.1. *(Unjustified accusation.) Assume that (with accusation p) A accuses B of β with scope S, and $C \in S$. Now accusation p is UNJUSTIFIED according to C if (i) C considers β NOT to express content that merits being the subject of an accusation from A to B, or otherwise (ii) EACH of the following criteria is met:*

(a) C considers β to be factually Inadequate, AND

(b1) C considers it NOT plausible to such an extent that β is factually adequate that C WOULD consider the accusation to constitute a reasonable instrument (from the perspective of A) for obtaining clarity on precisely that matter, AND:

(b2) C considers it NOT plausible that from the perspective of A it is plausible to such an extent that β is factually adequate that C WOULD consider the accusation p to constitute a reasonable instrument (from the perspective of A) for obtaining clarity (for A) on precisely that matter, AND

(c) C considers the virtue of the accusation in its capacity of a warning sign (to agents in S, but conceiving of β merely as a relevant pattern not necessarily related to A or to B, see "warning signal rationale") TOO LOW for C to hold that the accusation can be maintained even when factual adequacy of β is likely to be unwarranted or is unlikely to be properly investigated, AND

(d) WHILE C knows that β is both categorically unprovable and categorically irrefutable, it is NOT the case that C has so much more trust in A than in B that C considers it justified that A accuses B by way of an evidence immune accusation.

Using C as an arbiter of justification of accusations may be unreliable: if A motivates their accusation p against B along $(ii) - c$ with the argument that B is an enemy and that virtually any problem that is caused for B by A's accusation p constitutes an intended outcome of the accusation then C may go along with such considerations out of loyalty with A. It may even be the case that A has threatened C in order to go along with A's accusation of B.

7.1 Self-propelling accusations

An accusation (p in which A accuses B of β with scope S, and $C \in S$) may be unjustified according to the assessment of agent C but the verry existence of p may at the same time have three different side-effects:

(1) to produce an appeal to C to develop a degree of loyalty to A which creates room for justification (of p by C) on grounds $(ii) - c$ (with the understanding that loyalty to A turns the support of A in to a laudable act),

(2) and at the same time to develop a degree of trust in A and distrust in B which contributes to justification along the line of ground (d).

(3) S issues promises of their justification of p to other agents inside or outside scope S of p, with the result of a snowballing effect, so that increasingly more agents become involved by becoming aware of the existence of accusation p and by developing their own justification of p.

An accusation which generates its own support may be called self-propelling. Self propelling accusations are among the most powerful (and dangerous) speech acts available to a human agent.

7.2 Malicious accusations

Agent C may consider an accusation malicious.

Definition 7.2. *Agent C considers accusation p with accuser A, accusee B, accusable β, and scope S malicious if: (i) C considers p unjustified and (ii) C holds that the intention of A has been to create a self-propelling accusation.*

Computer programming does not provide obvious examples of unjustified accusations, self-propelling accusations or malicious accusations. There is much room for malicious promises, however, for instance in the context of phishing attacks.

8 Concluding remarks

The background of the absence of a side-effect in terms of the creation of obligations form promising lies in the initial analysis of Mark Burgess that inanimate agents can be autonomous and can engage in voluntary cooperation, while obligation is less plausible for an inanimate agent (see for example [15]). In a symmetric manner one may hold that accusations can play a complementary role as instruments for voluntary non-cooperation. However, in the context of computer programming we understand promising and accusation as means of communication of information in circumstances where it is non-obvious or even impossible to obtain reliable information regarding underlying facts.

We have introduced modal logic style notations for promise, threat, accusation and procusation. We consider this notation to be helpful for exposition, while we have no claim or expectation concerning the existence of useful, necessary, or sufficient axioms for promises and accusations.

An outline is given of accusations that may occur in the setting of computer programming. A simple conclusion can be draw: expecting a programmer A to provide a customer/user X with a program X in such a manner that no complaints or accusations are likely to come about requires formidable preparation by both A and B, as the space of possible accusations is enormous. Our account still underestimates the intrinsic conceptual difficulty of software engineering in the following sense: the assumption that a programming project starts with B providing a requirements specification S_{req} is often too optimistic. Instead prototype versions of X may be used as the best available expression of S_{req}, in which case it becomes almost impossible to organise an effective separation of concerns for A and B.

We have not embedded our discussion of promising and accusation in the interaction of programmer versus user/owner of programs in an institutional context. We hold that it is a characteristic of computer programming that third party assessments of the state of affairs are often hard to obtain. If the programmer role is played by a team fielded by a reputable international cooperation, of which there are many in computer programming, then it may be impossible to find a third party whom the task can be assigned to come to a reliable assessment of a problem situation. An unfinished program is unlike an unfinished building. Much more than in civil engineering the members of a programming team have in mind what must be done, and transfer of the project to another team may even be more costly than restarting a project, or even redefining the objectives of a project. We hold that these considerations justify our focus on a few key roles regarding computer programming.

References

[1] Annette Baier. Trust and antitrust. *Ethics*, **96** (2), pp 231–260, (1986).

[2] David A. Baldwin. Thinking about threats. *The Journal of Conflict Resolution*, 15 (1), 71–78, (1971).

[3] Jan A. Bergstra. *Promises and Threats by Asymmetric Nuclear-Weapon States.* χt Axis Press. ISBN: 978167318215, (2019).

[4] Jan A. Bergstra. Instruction sequence faults with formal change justification. *Scientific Annals of Computer Science*, **30** (2), pp. 105–166, (2020).

[5] Jan A. Bergstra. Promise theory as a tool for informaticians. *Transmathematica*, https://doi.org/10.36285/tm.35 (2020).

[6] Jan A. Bergstra. Promises in the context of humanoid robot morality. *International Journal of Robotic Engineering* , DOI: 10.35840/2631-5106/4126, https://vibgyorpublishers.org/content/ijre/ijre-5-026.pdf, (2020).

[7] Jan A. Bergstra. Qualifications of instruction sequence failures, faults and defects: dormant, effective, detected, temporary, and permanent. *Scientific Annals of Computer Science* **31** (1), pp. 1–50, (2021).

[8] Jan A. Bergstra. Defects and faults in algorithms, programs and instruction sequences. *Transmathematica*, https://doi.org/10.36285/tm.49 (2022).

[9] Jan Bergstra and Mark Burgess. *Promise Theory: Principles and Applications.* χt Axis Press. ISBN: 9781495437779, 2014; Second edition ISBN: 9781696578554, (2019).

[10] Jan Bergstra and Mark Burgess. Candidate software process flaws for the Boeing 737 Max MCAS algorithm and a risk for a proposed upgrade. https://arxiv.org/abs/2001.05690v1 [cs.CY] (2020).

[11] Jan Bergstra and Marcus Düwell. Accusation theory. *Transmathematica*, https://doi.org/10.36285/tm.61, (2021).

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

[13] J.A. Bergstra and C.A. Middelburg. Thread algebra for strategic interleaving. *Formal Aspects of Computing*, **19** (4), pp. 445–474, (2007).

[14] Bergstra, J.A., Ponse, A.: Execution architectures for program algebra. *Journal of Applied Logic*, **5** (1), pp. 170–192, (2004).

[15] Mark Burgess: An approach to understanding policy based on autonomy and voluntary cooperation. In: IFIP/IEEE 16th DSOM, LNCS 3775, pp. 97–108, (2005).

[16] Mark Burgess: Knowledge management and promises. LNCS 5637, pp. 95–107, (2009).

[17] Dag Elgesem: The modal logic of agency. *Nordic Journal of Philosophical Logic*, **2** (2), pp. 1–46, (1997).

[18] Gelperin, D., Hetzel, B.: The growth of software testing. *Communications of the ACM*, 31(6), 687-695, (1988).

[19] Andrew, J.I. Jones. : On the concept of trust. *Decision support systems*, **33** pp. 225–232, (2002).

[20] Besma Khaireddine, Ali Mili: Quantifying faultiness: what does it mean to have N faults? In: 2021 IEEE/ACM 9th International Conference on Formal Methods in Software Engineering, pp. 68–74, (2021).

[21] McIver, A., Morgan, C.: Correctness by construction for probabilistic programs. In: T. Margaria and B. Steffen (Eds.): ISoLA 2020, LNCS 12476, pp. 216–239, 2020. https://doi.org/10.1007/978-3-030-61362-4_12, (2020).

[22] Miller, R., Collins, C. T.: Acceptance testing. Proc. XPUniverse, p. 238. (2001).

[23] Pörn, I.: Some basic concepts of action. In: S. Stenlund (ed.), Logical Theory and Semantic Analysis. Reidel, Dordrecht, (1974).

[24] Williamson, T.: Modal science. Canadian Journal of Philosophy, **46** (4-5), pp.453-492, (2016).