Abstract

This thesis explores using a systemic view on software engineering to analyze its high failure rate. With the growing amount of bugs, and (emergent) misbehavior in software nowadays, it would pay to analyze the way in which we make this software first. Durability is proposed as a key property of development systems as a measurable indication of a system’s ability to prevent failure. Research into sustainable development is used as a methodic framework, both demonstrating how to find patterns of behaviour that have an effect on a system’s durability, and how to use these patterns to judge where to act to prevent potential failure. The method is demonstrated by several case studies, which show the analysis of durability as a useful method to critically analyze development processes and explicate patterns of positive and negative events.

Introduction

Software engineering, and most development in the digital world with it, has a high rate of failure; often being overtime, too expensive, or never delivering results at all. This failure is most often attributed to the way things are made - the development system - not so much the (properties of the) thing itself. Much research exists analyzing these failures from this point of view (cf. Kikuno 2005; Keil et al. 1998; Linberg 1999; Verner et al. 2008; El emam and Koru 2008, and many others), and has existed for a long time (cf. Abe et al. 1979). Despite this wealth of research, the high failure rate persists. This could have several reasons. Either the problem causing this the failure rate is unsolvable (or for that matter, undefinable), the problem changes with time, or our approaches used to define and solve it are too narrow in scope.

Computing is a comparatively young field. There are other, older, disciplines that also make things, and have significantly lower failure rates. To assume computing cannot learn from them, or refuses to do so, is tantamount to assuming young is synonymous with immature. Instead, the problem here must be that development in computing has some aspects that are unique to it. One easily demonstrated aspect which makes development in computing more difficult than, say, architecture is change.

Instead, our attention should be focused on finding properties of the system that relate to its ability to deal with change and on finding ways to “measure” it. A good starting point for such a property here is the capacity of a development system to endure (changes in) its environment, i.e. durability.

A durable development system is then easily defined as a system that continues to ensure the availability of its essential functions, thereby being able to fulfill its mission. In terms of dealing with change, durability implies the ability of a system to continue functioning in a changing environment - without too much compromise. But what is the essential function of a development system? It cannot be to merely make a thing\(^1\), Its mission should be to satisfy some needs\(^2\), by means of making a thing. The durability of a development system is therefore the measure to which it can continue to satisfy a number of, and potentially changing, needs.

Measuring durability is thus measuring the ability of a development system not to fail, by means of figuring out the (patterns of) events that lead to a decrease in durability. Durability can be calculated and reasoned with, but most work in this field (cf. Knight and Sullivan 2000) focuses on software as the system of interest. A development

\(^1\)Attacks, failures or accidents being events that arise during development, or due to changes in the environment.

\(^2\)Any development system with enough funds can attain such state of meaningless lastingness. This says little about the quality of those things, let alone what they contribute to the world.

\(^3\)Where a need can range from absolute must haves to things merely liked, as could be modeled with a method like MoScow prioritization (Clegg and Barker 2004).
system is of a significantly larger scope and complexity. As most methods require the input of all possible (not merely known) states and how a system reacts to them, it should be obvious that for a development system, a method to systematically gather those states, is first needed.

For this reason, it is necessary to clearly, and concisely define a way to gather that information. This thesis proposes a systematic way of doing so by both defining a mental model of the way we make things (an abstract development system), and adapting a method for analyzing sustainable development, which, apart from terminological difference, is quite strikingly similar in stated goal to ours:

“We must be able to recognize the presence or absence of sustainability, or of threats to sustainability, in the systems under our stewardship. We need proper indicators to provide this information, to tell us where we stand with respect to the goal of sustainability.” (Bossel 1999)

How we make things

When we speak of making things, we mean the physical objects that are intentionally designed and come about by our labor, more specifically to be called artifacts. Many ways of making artifacts exist in computing science. They can be categorized along a spectrum of strictness, ranging from the “Waterfall Model” approach inspired by (Royce 1970) favoring a strict linear development, to more iterative development like prototyping (Naumann et al. 1982), culminating in the Agile Manifesto (Beck et al. 2001) which sets out a way how to guide development without unnecessarily constricting it. However, even with such agreed-upon best practices, the exact ways in which artifacts are made still often differ on some points.

This is problematic. Firstly, someone looking at a development system, from the point of view of its method, has to have an in-depth understanding of that used method. They must have a mental model for it, a very exact - and preferably explicable understanding of both the development system and the artifact they are looking at. In a sense, they must have a formal model, of the development system, in their mind, and use it to structure their observations. It is simply not feasible for one, let alone multiple observers, to be intimately acquainted with the number of methods that one can come across. Secondly, most method-based views on development systems in computing are software-centric. While software is a usual endgoal, it is not the only thing we should be able to look at. Equally as important is the development of information in itself, or meta-information like data-structures. While problems are usually solved by implementing or adapting a piece of software, it is equally as important to be able to analyze a problem in the first place, both by systematically gathering the information in the first place, and structuring it in a way so it can be used for other solutions (i.e. a specification, or requirements document), like software.

For this reason, a more fundamental view on the development system is needed. If we abstract from the specific methods or paradigms used, we can look at the fundamental concepts and facts underlying all development systems. By doing so, we can create a system of generalities - which is not a claim to a general system - in line with General Systems Theory as described by (von Bertalanffy 1969).

We can create a starting point of such a system by reasoning backwards. On a very low level the common concepts to development can be seen as the artifact that was made, the need that it (hopefully) satisfies, and the means by which it was made. We also know that if an artifact exists, it was made, usually for a reason, which implies some needs. So:

A need can cause development, which can result in an artifact and an artifact implies development, which implies some (possibly unsatisfied) need.

The structure of, and relation between these concepts seem fundamental to a system of generalities for development. Artifacts are (hopefully) what they were made to be, but need and development have to be decomposed further. Luckily, need and development seem to have a common main systemic element: human beings. Both need and development can be seen as human beings undertaking some actions, in this case: acts of communication. It should therefore be clear that a system of generalities for development should adopt a communicative view and reason with human beings as its main elements.

Such a view can be found in a paper on the “Second-Order Information System” (Hoppenbrouwers and Proper 2004). It describes development as a second-order system that creates first-order systems (artifacts) by means of human acts of communication, or more specifically, negotiation. While only being a sketch, it provides a starting point for some common concepts and facts for our development system. In such communication, three kinds of statements should be distinguished; being assertions, modifications or retractions, which result in the creation of texts. These can be implicit (in the minds of people), or explicit (recorded verbalizations or transcriptions). Both need and development can be described in terms of these simple actions. Need is the act of actors explicating their implicit needs by stating what they need. Development is the communication of actors on how to satisfy those needs, and, hopefully, doing it after reaching consensus on how to do it.

While we now have the means to describe, on a fundamental level, how the concepts and facts of the development system work, we do not yet describe, in enough detail, the relation between the artifact made and the needs stated. For this, we can use the Rationality Square (Wupper et al. 2009). It reasons that a description of properties is used to construct a blueprint, which is used to build an artifact in physical reality, which then should have the described properties in physical reality. In computing terms, implicit needs are made explicit needs in the form of a specification, according to which source-code is created, which is compiled to create the program in physical reality (i.e. physical memory).

The program, at that moment, should satisfy the needs it was built for. However, those needs can, and likely will, change over time. This requires the development system to stay active, to maintain and further develop the artifact, or perhaps create a completely new artifact to satisfy the new needs. It should stay active as well, to deal with possible problems arising from emergent (mis)behavior that could conflict with the artifact satisfying the needs it was made for (Mogul 2006).

4Our understanding of survivability is essentially a form of sustainability. However, with ongoing discussion on what exactly constitutes sustainability, and whether it is useful at all (Pearce et al. 1989; Farrel and Hart 1998; Sutton 2004; Beckerman 1994), it was thought to be wiser to start from a less controversial definition.

5However, we are not interested in the moral aspects of ensuring durability of our systems, as we do not necessarily see ourselves as stewards of our development systems.

6While (partial) automation of software development is possible, such methods are not mature enough to fully rely on, nor could they entirely replace the human factor in a software development system.

7Note that these would still all be assertions from an observer’s point of view.
These understandings form the basis for our fundamental development system. A syntactical formalization is given in Figure 1 to serve as a compact lexicon, and as an explication of how we look at development. It necessarily includes some simplifications. In general, most of these are reduction of complex things, like communication, to either black boxes or simplified interactions. This is done so as not to lose focus on what this work is about. It would be counterproductive to increase the decomposition, and thereby the complexity of the model, with little to no direct benefits for the application of the system in this work. Nevertheless, some simplifications are done with a specific thought in mind.

The model does not explicitly discern between direct, and indirect communication. While they are subtly different (Coulmas 1986), especially in the context of development (Pruitt 1971), both modes of communication can still be modeled in the system. This is done by assuming that in development, most indirect communication will be based around figuring out what an actor’s needs are without that actor explicating them. This can be done by modeling such an unnamed actor, and instantiating some of their hinting, suggesting or insinuating mannerisms as statements (Holcroft 1976) prescribed to them, by another actor.

It also does not include any notions of compilers, integrated development environments or other (personal) computer specific tools. This is done for a specific reason. As computing is about computation, not so much the computer (Denning 2007), it would be counterproductive for a fundamental model to restrict itself by explicitly including specific means of computation. Furthermore, there is no reason to assume that computer always has to be synonymous with what is understood as today’s personal computers, instead of any other thing that can compute. Weizenbaum understood this, and reasoned in his book on “Computer Power and Human Reason” that such, perhaps dogmatic views are good for very little progress:

“Yes, the computer did arrive ‘just in time’. But in time for what? In time to save - and save very nearly intact, indeed, to entrench and stabilize - social and political structures that otherwise might have been either radically renovated or allowed to totter under the demands that were sure to be made on them.” (Weizenbaum 1979)

Furthermore, in the context of failure in development, it is again not problematic to not heavily, and explicitly reason about technical tools and their application. Most project failures can be traced back to events that happened before technical actions were carried out. Indeed, according to literature, most (signs of) project failures are:

“determined before a design is developed\(^8\) or a line of code is written” (Reel 1999)

**Reasoning about durability**

As stated earlier, few to none methods exist in computing for gathering durability data on development systems, instead of the things they make - their artifacts. There is, however, a wealth of literature and methods, from other fields of study, for gathering data on a related property - sustainability, ranging from approaches that are problem-specific, amongst others (Lopez-Ridaura et al. 2002; Richardson 2005; Nijkamp and Vreeker 2000; Alberti 1996; Vanegas 2003; Lewis et al. 1997) to more general approaches (Turner et al. 2003; Ugliati and Brown 1998; Bossel 1999; Fiksel 2006) and many more. This work will adopt Bossel’s approach described in “Indicators for Sustainable Development: Theory, Method, Applications” (Bossel 1999). It is broadly applicable, and built upon further in literature (cf. Moffatt 2001), making it especially easy to adapt for our purposes.

Bossel’s method gathers data on sustainability by systematically defining indicators that describe the satisfaction of a number of orientors, which influence the sustainability of a system. This is done recursively for a number of basic orientors and subsystems. The given basic orientors are argued by Bossel to be an exhaustive and non-reducible set of properties that ensure sustainability, meaning that sustainability is assured, when none of the orientors are left explicitly unsatisfied.

\(^8\)“Developing a design” should be understood as writing a specification, the process of explicating needs and recording them. It should not be mistaken to mean that most failures occur before there were ever any (implicit) thoughts about a design.
These orientors effectively cover durability too\(^9\), so as a starting point for our analysis we will only need to understand the orientors in our context, and define our subsystems. Bossel’s description of the basic orientors defines them as the ability of a system to:

- physically exist in its normal environmental state (existence)
- effectively harvest necessary resources (effectiveness)
- freely respond to, and cope with environmental variety (freedom of action)
- protect itself from unpredictable threats, i.e. environmental variability (security)
- adapt to changes in the environment, learn, adapt and self-organize (adaptability)
- interact productively with other systems (co-existence)
- deal with special needs of sentient subsystems that require identity, love, affection, avoidance of pain and stress (psychological needs)

However, those descriptions are not clear enough for our context, nor can they be easily used as guidelines for (domain) experts to reason about their development. To ensure that they can, and the orientors are consistently interpreted, our understanding of their them is given below as the ability of a development system to:

- possess the minimal amount of needed systemic elements (i.e. actors, tools), and to retain those elements, even in face of organizational regulation negatively affecting their availability
- explicate (implicit) ideas and thoughts (i.e. communication), influence those ideas and thoughts (i.e. negotiation), and accurately refine them into specifications and blueprints
- be able to employ new methods or approaches, and not be limited in that capability by organizational or cultural dogma
- be able to withstand sudden (temporary) internal changes or limited systemic effectiveness, (i.e. unavailability of an actor or tool), by means of redundancy
- be able to argue for, and bring into effect, needed changes in its internal elements (i.e. education of actors) to deal with new environmental elements that require such changes (i.e. adaption of a new tool, operating in a different cultural/social context)
- both have the capacity to - effectively - communicate with other (development) systems (i.e. share knowledge with others), and to ensure that produced artifacts (i.e. specifications, blueprints, artifacts) are made in such a way as to be exchangeable with other (development) systems (i.e. avoidance of proprietary methods/tools)
- deal with psychological status of its sentient systemic elements (i.e. actors) by promoting means that do not cause excess stress, neglect, apathy or lack of respect to such sentient beings

\(^9\)Consider that our definition of durability is, essentially, a subset of, or limited view on sustainability, and therefore, any system which is deemed to be (fully) sustainable is necessarily also durable.

Bossel’s orientors (Bossel 1999) applied to our topic, development in computing

With the basic orientors now given, a decomposition of our development system into relevant subsystems is needed. A useful decomposition is both compact, and can relate its subsystems to distinct things in the real world. In other words: subsystems should be chosen in such a way that with the results of a durability analysis, they can be pointed at (and blamed or celebrated) in the real world. For development in computing, a high level decomposition to achieve this would discriminate between things originating from human factors, technological factors, and organizational factors. We define them as being:

- human, the collection of human beings and their (inter)actions, i.e. (properties of) actors, communication and negotiation
- technology, the collection of tools and their inherent properties, i.e. (properties of) development environments, compilers, and their suitability or usefulness for a given case
- organization, the collection of regulatory objects that influence and prescribe behavior of human and technology elements (not necessarily consistent or satisfactory), both things that have to be obeyed (i.e. budget and deadline), and things that ought to be obeyed, (i.e. principles and guidelines).

**Our basic decomposition of development in computing**

These classifications are orthogonal to each other, resulting in a matrix of at most 21 orientor-subsystem combinations (see Table 1), capable of fully describing the durability of a development system.

<table>
<thead>
<tr>
<th>orientor/subsystem</th>
<th>human</th>
<th>technology</th>
<th>organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>existence</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>effectiveness</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>freedom of action</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>security</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adaptability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>co-existence</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>psychological needs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The orientor-subsystem indicator matrix

**Finding indicators**

The act of finding indicators and concluding a system’s durability from them is a fairly simple iterative process. When a case has been modeled in terms of our development system (i.e. instantiated), it should be analyzed chronologically. Defining indicators is done by going through the development from beginning to end or present, finding events that correlate with the basic orientors, assigning them to a subsystem, and finally denoting their satisfaction as positive or negative. Not all combinations will be meaningful, as for instance technological assets rarely have a measure of sentence, making their psychological needs a pointless thing to reason about. With this set constructed, weighting should be applied to calculate scores of each orientor and subsystem’s satisfaction. Durability can then be simply computed by making sure that no orientor has a score that is smaller than zero, while also acknowledging that orientors with a score of zero can at most only be seen as durable on short-term. Individual subsystems of an orientor can have a score smaller than zero, provided they are compensated for by (over)satisfaction of a different subsystem.

**Pitfalls**

As simple as the method is, there are many pitfalls that should be avoided. A resulting set of indicators could, for instance, lead to pointing at a specific orientor that was left significantly unsatisfied, yet it would be unwise to directly assume this was the cause of the
problem. A more holistic look at such results is needed, as (un)satisfied orientors usually are so because of a cascade of (dis)satisfaction of things before them. Instead of focusing on single orientors, care should be taken to find correlated events, or chains of satisfaction, to trace back such cascades as far as possible. A system that is deemed not durable because of unsatisfied psychological needs could for instance actually be caused by too strong satisfaction on security, which limits the freedom of action of its actors, which then in turn causes those unsatisfied psychological needs. Knowing this, attempting to “repair” the durability of this system would be in toning down the satisfaction of the security orientor, not the psychological needs that were left unsatisfied as a result of them. In conclusion, care must be taken to differentiate between causes, and symptoms.

Differentiating between artifact and development system when finding indicators must also be kept in mind. While the development system encompasses its artifact(s), indicators that explicitly reason about the durability of an artifact, give little useful information about the durability of its creator. To illustrate, to satisfy the orientor existence we quickly come to intuitive indicators like “the artifact should be able to run on that computer”, but such a term muddles this exact difference. It is the development system’s task to ensure such things as stated above, indeed, but we should reason from the development system’s point of view, defining an indicator like “specifiers are familiar with the deployment environment”, and from there assume that the artifact will be made to run on the intended computer(s).

Another problematic step is determining the relative weight of each indicator. It should be clear that some events are more significant than others; if a main domain expert quits and takes all his knowledge with him, it is demonstratively worse than the psychological effects of everyone having to wear silly uniforms - but quantifying that relative significance is difficult to do. It cannot be done a priori, nor can it be done universally, making it quite a “wicked problem” (Proper 2003). This necessitates extensive case studies for each cultural area and period (i.e. development in India would be different than in America, and depending on the era, their zeitgeist would be too), instead of attempting to inductively define such sets beforehand.

Quantification of indicators themselves is also a pitfall. Attempting to strongly formalize them to improve their objectivity and usefulness for calculations, results in more practical problems than it solves. Each indicator should be clear as to what it measures, or reasons about, but this can be achieved without strong reliance on formalizations. Compare the following indicators attempting to reason that “there are enough programmers”:

1. a project has to have at least 2 actors of type coder with so-and-so much time available
2. >=1 coder with >= 1.0 FTE
3. \forall x \in \text{projects}, \exists a \in \text{actors}, \exists b \in \text{actors}, \text{involvedIn a x } \land \text{involvedIn b x } \land a=b \land \text{ofType a coder } \land \text{ofType b coder } \land \text{withInvolvement a 1 } \land \text{withInvolvement 1 }.

The first indicator will make sense to anyone. The second and third will only make sense to an increasingly smaller public. While it is a given that the third indicator has less possible ambiguity than those before it, it will prove of little use if people do not know the syntax, or worse, only barely know the syntax and misinterpret its meaning. While it is generally enough to paraphrase such formalisms in natural language or demonstrate their consequences (Hall 1990), doing so will result in the first indicator again. As that one is sufficient for our uses, this work will forego strong formalization of indicators, and instead structure indicators according to a compact lexicon (i.e. our model of a development system).

To summarize, some common pitfalls that should be avoided are:

- Stopping analysis too soon by confusing symptoms of (lack of) durability with their causes
- Confusing the artifact with its development system
- Attempting to strongly formalize indicators too soon, or attempt to quantify their relative impact on durability

Case study

This case study has several purposes. First, and foremost, to study the usefulness of the method in itself, and secondly to make a first start in collecting possible common indicators, that could be used to create more general sets of durability indicators. To keep the case study manageable within the scope of this work three cases were selected. The cases were selected based on their different end goals and methods they used to achieve them. This ensures there should be a sufficiently diverse amount of data when analyses are compared. The cases include a project which aimed to deploy an existing artifact in a new environment, a project which aimed to define an abstract data-structure, and a project which developed a new artifact from scratch.

The data needed to deduce indicators was found by means of several semi-structured interviews. These interviews were used to clarify the (chronological) development progress, and their respective notable milestones. The interviews were held in a direct and accusative way, as to provoke strong (emotional, non-verbal) responses. These responses served to verify critical moments in development, as they were related to decisions that proved to be wrong or ineffective ways of doing things later on. Results were presented to, and agreed upon by the respective interviewees. For professional purposes, personal details and information that could identify people or organizations involved have been anonymized.

In the following case studies, results are presented as a radar chart which demonstrates the strong points and weaknesses of each subsystem, per orientor. The method used is shown here, with a list of events and processed numerical data available in the appendix.

1. Gather information about development, for instance by interviews with several stakeholders.
2. Create a chronology of the development (process).
3. Indicate (significant) events in the chronology.
   3.1. Define indicator(s) for the event and relate them to a specific orientor and subsystem.
   3.2. Assign a score to the indicator (respectively +1 and -1 for positive and negative indicators).
4. Summate the scores per orientor, per subsystem

Our method used for gathering and processing data

Case 1: Deployment of an existing artifact

In this case, the initial contractor wanted to implement an existing artifact in his organization. The artifact at hand, a content management system optimized for educational content, was already in use by several other, similar, organizations. He learned of it through both active cooperation, and regular exchange of ideas with those organizations. Seeing the use of it in those environments, and reasoning that his organization was similar in goals
and its employees’ needs, he decided to implement it in his own organization.

+ co-existence, organization (promotion of cooperation with other organizations’ development systems)

The needs were to have a means to facilitate a move to peer-based creation and review of educational materials. Simply put, an artifact was needed, that enabled and promoted users to create (educational) content in it, share it with others, and refine it by actively promoting multi-disciplinary cross-fertilization and peer-review. The problem here is that those needs were attributed to the proposed end-users, which meant that the development system should not only create an artifact capable of satisfying those needs, but also ensure that it is used - and perhaps more importantly accepted - by those proposed end-users.

<table>
<thead>
<tr>
<th>actor(s)/role</th>
<th>contractor</th>
<th>expert</th>
<th>specifier</th>
<th>coder</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Tiwi Consultants</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 2: Case 1 actors and roles

An overview of the major actors of this case is shown in Table 2. While all necessary roles are accounted for, it should be noted that there is a strong reliance on John to perform several roles, and that there is only one coder. As the company that developed the artifact kept it closed-source, it became not only the sole coder - but ensured no other coders could realistically work on the artifact. Another possible problem was the reliance on several, both internal and external, consultants, who were not necessarily paid or contracted employees, and could thus not be fully relied upon to always be available.

− security, human (no possible redundancy of coder role)

Development, or more accurately deployment and adaptation, began in 2002. Before this time, John lobbied unsuccessfully to his superiors to deploy the artifact. However, with the growing interest in it, in due time eventually a group of like-minded people from different parts of his organization helped him successfully convince his superiors to implement the artifact. While potentially frustrating to the contractor, this delay enabled him to already secure a small target audience, sharing the same interests, who agreed to work with - and give feedback on - the artifact once it was implemented.

After John’s organization green-lighted the implementation, several months were spent negotiating with the artifact’s creator - Tiwi - on costs, possibilities to personalize the artifact, and more technical details like the location of the artifact. Eventually a contract was agreed upon - not in small part due to help from another department in John’s organization who provided the technical know-how for these negotiations. John’s superiors eventually provided the necessary financial means for both starting development and continuing it, but soon thereafter lowered it beneath the amount needed, and agreed upon in the contract. While this could have stopped development right there and then, John, driven by his strong belief in the use of the artifact, used his personal assets to cover the remaining financial means needed.

+ co-existence, human (inter-organizational cooperation)
− existence, organization (needed financial means approved and subsequently lowered beneath minimal threshold)
− security, human (existence of development system completely dependent on single actor)
− psychological needs, human (significant pressure and stress to have system succeed because of strong personal involvement)

Subsequently, the artifact was implemented relatively smoothly. For the first months, John personally interacted with his (still small) target audience to refine the artifact, all the while attempting to increase his target audience. The latter proved to be a difficult task, as convincing other groups of intended end-users of the use, or simply having them agree to test it was a cumbersome process with little results to show for. Most of the target audience was unwilling to use the artifact, not because of a principal disagreement with its goals, but because few wanted to be the first to semi-publicly release their educational materials.

Furthermore, they proved reluctant to use yet another information system, because of negative experiences with (organizationally mandated) systems in the past, and the required time investments were seen to be too large. Of all these problems, only the last one was solved in some way, at another sacrifice of John’s own time and resources.

A small group of additional end-users was convinced to use the artifact, have their materials in it, provided that John or his consultants would take care of the data-entry, makeup and any subsequent needed changes. This was done for a small number of end-users, with the hope it would convince them of the usefulness of the artifact, but such convictions failed to come in the, by now years, John had now spent on implementing the artifact.

− effectiveness, human (insignificant amount of end-users restrict possible amount and variety of necessary feedback)

At this point, several years later, John was one of the few remaining active actors in the development system. No significant amount of new end-users had been acquired, still remaining afraid, uninterested or stubbornly refusing to cooperate. By now most external consultants were no longer an active part of the project, whether due to lack of financial means or a lack of things to consult about, and John became essentially the sole remaining developer and organizational salesman, excluding the external coder. Eventually, the needs he once tried so hard to convey to his target audience, were let go, and little more was done than keeping the artifact alive in its current state.

− effectiveness, human (static amount of end-users and developers eventually stop feedback entirely)
− psychological needs, human (no acceptance, nor recognition of the work done given by end-users or organization)

Interestingly enough this was not the end of the development system. Through a combination of growing interest in digital testing (i.e. a new needed property), and cooperation with another department, John became aware of a new possible use for the artifact. As there was a strong target audience now for the new needed property, John managed to adapt the artifact over the course of several months to fit in with the current situation and change its goal to become both a means for digital testing, and a database for the questions needed for them. This proved to be a success, and in a matter of months John managed to achieve what he did not manage before; a large target audience and active use of the artifact, albeit for a completely different needed property.

+ adaptability, human (accepting new needs and figuring out how to adapt the artifact to accommodate them)
+ freedom of action, organization (organizational apathy led to no restrictions in changing development goals)
In the end, the artifact ensured its usefulness and place in the organization by serving both as a tool for the new needed artifact, digital testing, and by already having the means to function as a database for the assets needed for the tests. Perhaps as a lesson from the past, end-users were able to supply the needed assets in their own most used languages and file types (i.e. LaTeX, Word), which the artifact had been made to work with.

+ co-existence, technology (artifact uses standardized input data)

An overview of the indicators converted to numerical scores for this case is shown in Figure 2. Their implications, relations and effect on the development system’s durability are discussed below.

![Figure 2: Case 1 orientor satisfaction per subsystem](image)

Perhaps remarkably, the organization subsystem seems to be neutral in terms of satisfaction, while the human subsystem is left significantly unsatisfied. As dissatisfaction of (human) actors can often be caused by negative working conditions, this combination of results seems strange at first. However, the explanation here can be found in the key description of the organizational behavior in this case: apathy. Most organizational indicators can be explained in the light of it; the budget was reduced because of a lack of interest, yet the actors were left (significantly) free in their actions. This causes the responsibility for making the project a success, or durable, to be completely that of the actors themselves.

In this case, that was also the main problem. The main actor attempted to make the project a success, going as far as using personal resources to level the playing field - certainly increasing the pressure on him - which caused other problems. While the dissatisfaction of the existence orientor was resolved with this course of action, it had significant implications for others. The security orientor here became strongly unsatisfied because the mere existence of the development system now relied upon that sole actor, which made any amount of redundancy - actor wise - both non-existent and impossible.

Effectiveness was unsatisfied, because of a problem already signaled earlier - a lack of basis or target audience. This, while not necessarily a problem when mainly developing an artifact and then forcing people to use it (whether through organizational means, or having the artifact be a defacto standard), does not work when the artifact is meant to be tailored to a specific environment, yet there is simply no feedback to be gotten from it. The additional problem of such gambles is its eventual negative effect on the psychological needs of the related actors, and specifically the one who defined the target audience. In this case this was aggravated because of the other factors negatively contributing to psychological needs, making the risk of the actor giving up (and the development system collapsing with him) even larger.

Intuitively the development system might seem to have restored itself - a target audience was found and a useful artifact was made, after all - but it must be noted that the unsatisfied orientors, with the exception of effectivity, still remain unsatisfied. While on the short-term development might continue, eventually the system should be expected to collapse because of the lack of actor redundancy.

Concluding, what this case exemplifies above all, is that attempting to fix a problem by addressing its symptoms (i.e. lack of budget), instead of its causes (i.e. organizational apathy) rarely ever works out in the long term.

**Case 2: Definition of an abstract data-structure**

The needed property in this case is a bit more complex. Eventually, the goal is an abstract data-structure for an encyclopedia of a specified field in the humanities. The information in it, nor the physical realization of the encyclopedia are the goals, but the underlying structure is. Complicating things, the domain experts already realize, and agree, that such a project can only succeed through direct experimentation. Because of this the needed property the development system must fulfill, is the need for an environment in which this process of experimentation and verification can be both be facilitated, and actively stimulated.

![actor(s)/role](image)

<table>
<thead>
<tr>
<th>actor(s)/role</th>
<th>contractor</th>
<th>expert</th>
<th>specifier</th>
<th>coder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dick</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Harry</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>John</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Oesi (initially)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Case 2 actors and roles

An overview of the main actors involved and their respective roles is shown in Table 3. Tom, Dick and Harry are all well-versed researchers in their field, but have little knowledge or expertise concerning technological solutions. The specifiers and coders (or, implementor of existing code) shown were mutually exclusive, as they were not part of the development system at the same time. As will be explained later, a change of specifier and coder was made, after which Oesi was no longer involved in the project, nor a part of the development system.

+ effectiveness, human (variety and consistent availability of experts)

Initially, Tom was developing a new piece of software with Oesi, partially because he was unaware of other options, partially because this actor was the defacto coder to go to in his organization. Development went nowhere, as Oesi required exhaustive specifications and exact descriptions of what technology was needed - which could obviously not be expected from Tom, Dick and Harry - let alone most domain experts in any case. Eventually, after a significant amount of development time with little progress to show for, Tom, by now acquainted with John and aware of his other options, made the decision to stop cooperation with Oesi and tried having development guided by John. Co-incidentally, this move is characteristic of the current trend away from stricter, hierarchical development methods and towards more agile ones.

+ adaptability, human (contractor becomes aware of, and stops cooperation with ineffective coder(s))\textsuperscript{10}

\textsuperscript{10}Please note that this supersedes earlier possible negative orientors, as there were no lasting negative effects from the earlier (period of) development.
All actors now aware of, and agreed upon the need for an agile approach, John spent much time figuring out what the domain experts wanted, and in what ways what they wanted had already been realized before. Eventually, parallels with another encyclopedia project, Wikipedia, were drawn, and both expert and specifier/coder now shared a similar vision for the needed artifact. John suggested, and implemented the software used and developed for Wikipedia’s purpose, to move towards a first tool in which the experts could attempt to create an environment that would facilitate the creation of the needed data-structure.

+ co-existence, technology ((re-)using open-source components for artifacts)

The eventual encyclopedia, and the data-structure with it, had to cater for both researchers and educators and their respective needs. This, combined with the direct experimentation approach, meant that research and education had to come together - but Tom, because of political matters, had little to no education to bring together yet. John stepped in and offered to test the artifact in a course he taught, which eventually proved both successful not only for this case, but spawned other development (systems) with similar ideological goals too. By now the groundwork for experimentation was in place, and Tom and the other domain experts could use the artifact to experiment with possible data-structures.

+ co-existence, human (actors can use each other’s expertise and background to refine ideas)

By now the development system did not fulfill its mission entirely yet, as the data-structure had not been created, nor explicitly agreed upon by the experts yet, meaning that possible development in terms of adapting the artifact (i.e. altering or implementing new functionalities) could still be necessary. Nonetheless, it is remarkable that development, and reaching the end-goal, seemed to go faster and more effective with this far more free and open way of development, as opposed to the strict guidance Oesi demanded earlier. During the following years, the experts used the artifact to experiment with different possible data-structures, while exchanging ideas and feedback. This process quite simply created, and refined small data-structures - while they were being used and immediately verifying whether they were useful with the end-users. As such, it benefited immensely from the large and varied amount of feedback from both the experts and other researchers using the artifact by now. John supplemented this with additional feedback and experiences with the same artifact in a different environment, which led to the experts coming closer and closer to their ideal data-structure.

+ existence, technology (adaptation of existing artifact as a means to develop data-structure in)
+ effectiveness, human (variety in actors and environment causes large variety of, and capacity for, feedback)

Eventually, both Tom and John’s experimentation reached an equilibrium where few new ideas or changes were introduced. By now, time and the artifact found by John had allowed for the methodic and careful experimental creation of the data-structures, which by then were directly in use by the encyclopedia itself. With the initially stated goals reached, and the development system still quite intact, the experts continued experimenting with other functionalities and possibilities the artifact had. This eventually resulted in the implementation of more functionalities that were not part of the initial needs, like journals and improved communication protocols within their group. Presently, the development system, having long fulfilled its initial goals, is still active and working towards new possibilities, ranging from small day-to-day improvements, to experimenting with building blocks for new, yet unneeded functionality for the future.

+ freedom of action, organization (no given deadlines or required milestones)
+ freedom of action, human (envisioning further applications for the artifact, i.e. open-minded approach)

Nonetheless, the most significant results from this case seem to be the satisfaction of co-existence (both for the technology, and human subsystems) being correlated with strong(er) satisfaction of the effectiveness orientor. With the experimentation both able to start soon in development (because of the use of already existing artifacts, i.e. co-existence), and the large amount of people involved in the artifact (because of the cooperation between actors, i.e. co-existence), there was a constant process of attempting new things, and receiving feedback from a multitude of users (i.e. effectiveness).

Concluding, what this case can teach us, is that agile approaches definitely do not have to be slower in delivering useful results - especially if the domain experts are not technological experts. While the lack of strong deadlines certainly played a role in making this system so durable, it could also be said that the agile approach, combined with the use of ready-made free (both of costs and rights) software, played a significant role in its durability, by ensuring that the work (and thereby time-investments) were balanced over all the actors and could be focused almost solely on experimenting with the data-structure, instead of fruitlessly implementing new artifacts for it.

Case 3: Development of a new of artifact

In this case, a new artifact was developed to simplify several processes. The prior situation was that of several programs and databases which, while semantically related, had no ability to connect or communicate with each
other. This resulted in an inefficient way of working for its end-users as the programs were not unified interface or usability wise, and when data had to be transferred from one program or database to the other, it had to be done manually. The needed property was thus to be able to perform the same functionalities as the existing set of artifacts, but in a single environment, without needing to manually transfer data from point to point again.

<table>
<thead>
<tr>
<th>actor(s)/role</th>
<th>contractor</th>
<th>expert</th>
<th>specifier</th>
<th>coder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dept. Director</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Dept. Office</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Coders</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Case 3 actors and roles

The major actors are shown in Table 4. John was an end-user in the prior situation, and took it upon himself to improve that situation. With no explicit backing from his organization, he started the development himself, fulfilling essentially each role in development, which obviously could present a problem later on. The main domain experts John could rely on for other views on what the new solution should be were his department’s director, and office. Later on other domain experts from within his organization would give incidental advice, but played a less significant role in development. Another pair of actors temporarily fulfilled the coder role at a later point in development, but did not last as a part of the development system.

- security, human (complete reliance on single actor)

Development began when John, not set on creating a new artifact yet, attempted to improve the prior situation’s effectivity by connecting the existing programs and databases to each other. This did not work out, both because (technical) documentation needed for such ventures was lacking, and because strategic incompatibilities from his organization gave him little time to work on it.

- co-existence, technology (using existing artifacts impossible for lack of documentation or proprietary programs)
- existence, organization (apathy towards needed property results in lack of development time)

However, even with these problems and a lack of time, money or interest, John eventually managed to create a prototype. As there was no (known) way of interfacing with the existing databases to retrieve the needed information, shadow databases (i.e. redundant copies of the existing databases) were created as a temporary means of having the information available.

- security, technology (artifact dependent on being able to create shadow databases)

With this, and some feedback from his domain experts, mostly on personal title, he created an artifact which was used to demonstrate the usefulness of both his needs, and the way he had implemented them (i.e. the artifact). This worked out in a sense, as the system was appreciated for what it did, and used by the people it was targeted at, but also made it all the more problematic that it relied on the integrity and continued availability of the shadow databases. John’s department eventually offered him the services of a programmer, but the accompanying demands, an exhaustive specification, essentially prohibited this.

- existence, organization (insignificant amount of end-users restrict possible amount and variety of necessary feedback)

On a positive note, the artifact was well received amongst John’s peers and gained a solid user base in his department. Its usage grew, and the amount of feedback with it, but the development system did not. John was still the main specifier and coder, and his agile method of development became increasingly incompatible with his organization’s old-fashioned, hierarchical methods of development.

- effectiveness, human (variety in environment causes large variety of, and capacity for, feedback)
- adaptability, organization (incapable of adapting to other development methods, the organization could offer little help to John)

Over the following years the artifact was used, and slightly updated where necessary, but many problems started to appear. There were several key events that added to this erosion: an end-user tasked with filling in specific information falling away and nobody else doing it any longer; the department’s office being overworked and reducing the amount of functionality they used, which other users grew accustomed to and no longer expected to function; the department’s direction lack of explicitly stating the need for the use of parts of the artifact - and feeling it was not their problem to keep it working. Such events caused an erosion of the parts of the artifact that were used, which also affected the development system in terms of feedback it still received, and the means (i.e. time and money) their developers could negotiate.

- effectiveness, human (reduced feedback little to no explicit feedback on functionality that is no longer used, making other ways of satisfying its underlying needs impractical)

The final destructive event came in terms of two coders, pushed forward by the organization, who, in contrast with John’s wishes, changed the development environment to one that was both less agile, and not familiar to him. A switch of programming language and further work was also planned, but never completed when the programmers left the development prematurely. There were now two versions of the artifact, each with their respective development environment and needed tools and skills for maintenance. With his version already outdated, and lacking the skills needed to further develop the other version, his capability to make alterations to the artifact was severely compromised. From this point on, partially because of a lack of motivation and not being able to use the tools, John stopped active development.

- freedom of action, human (actor compromised in ability to develop by unwanted switch of development environment)
- psychological needs, human (actor’s wishes ignored about development environment change)
- adaptability, human (actor unwilling and incapable of adjusting to new development environment)

The artifact, by now set in stone, was no longer actively developed or maintained. As development stopped, so did most of its usage, until eventually little remained apart from those bits and pieces that were integrated into other systems, which, as a final positive note, was possible because of the open nature of the created artifact.

- existence, human (no actor with role contractor, specifier or coder any more)
+ co-existence, technology (parts of the artifact capable of being integrated into other artifacts\(^{11}\))

\(^{11}\)Strictly speaking, only parts of a blueprint can be integrated into another blueprint, and thereby become part of another artifact, but we assume the reader to understand this shorthand notation.
It should be clear from the collapse of the development system, that it was not durable. Figure 4 gives an overview of the results.

Figure 4: Case 3 orientor satisfaction per subsystem

A dissatisfaction of most basic orientors and all subsystems is clear, but from the overview of development it is also clear these were not solitary events. From the beginning the lack of organizational cooperation or reservation of means for development (i.e. time and money) caused the need for the actors to improvise, which in turn caused the artifact to rely on external databases, creating another problem with the security orientor. It is also somewhat obvious to rely on external databases, creating another need for the actors to improvise, which in turn caused the lack of organizational cooperation or reservation of means for development (i.e. time and money) caused the organization to act like a disjointed entity.

Could it have been prevented? Perhaps if there had been greater actor redundancy, then certainly the above cascade could have been prevented. But that ignores the other problem this case exemplifies above anything else: that if the organization or environment is incompatible with the development system’s method, one or the other will have to change, or eventually give up due to all the friction and stress such incompatibilities cause. Therefore, if any top-level cause for this development’s lack of durability should be named, it would have to be the lack of adaptability of the organization. Thus, attempting to make this development system durable should be done by making the organization more adaptable - a thing far out of the scope of the actors involved in this development.

Concluding, what this case can teach us is that attempting to make things, according to some method in an organization or environment that is used to doing it by a different method will create friction and frustration - unless both parties can come to a mutual understanding of what the best method is for that development. If they cannot, or either of them refuses to budge, the development system is not durable on the long-term no matter what.

Discussion

There are some important aspects of the method that the case study brought forwards, both positive and negative. Judging from the cases we studied, and the assumption that further case studies would corroborate our impression, we believe that (1) it is a simple method in its use, especially compared to other, more technical methods. In our experience, case analyses require little extra (technical) skills or knowledge apart from understanding the orientors as applied to development, and how to look at a development system. While our method uses few formalizations and relies on natural language to express its results - which at times may seem comparable to those of intuitive reasoning, we do so in a reproducible, systemic way. Because of this our method is easier to wield than other approaches that favor strong formalization and mathematical approaches, like the formal approach shown by (Knight and Sullivan 2000). Therefore we feel that our method is a better candidate for widespread use - for instance any project manager wishing to ensure the durability of his development systems, as it does not presuppose or explicitly demands years of training in computing science or technical skills.

(2) There are common causes for a lack of durability - or failure. The case study showed that some (cascades of) orientor satisfaction occurred in more than one specific case, hinting at the generalizability of such factors. Some of these problems we found were organizational apathy or dissonance and a (too) strong focus on (indirectly) satisfying some orientors, by which other related orientors were often left ignored or otherwise affected.

Organizational apathy was shown in different ways in case 1 and 3, in the former manifesting as a complete lack of interest and regulation of the development, and in the latter as the absence of acts that explicaited their sense of responsibility for the development. Both cases were affected by this - positive in an increased amount of freedom to act - but also quickly realizing there would be few safety nets to catch them, or other actors to share responsibility for things gone awry. The problem here then became the feeling of the developers that they were entirely responsible - manifesting either as increased stress and pressure not to fail, or increased stress and frustration at the organization not caring about his performance. Thus it seems that some involvement and responsibility of an actor’s organization is necessary to prevent such psychological issues.

Organizational dissonance is any mismatch between the attitude, or favored culture of some organization, and those of the actors in the development system. The most obvious effect of this was an inefficient use of means - or lack thereof, as the tools or other actors needed by the developers were often not, or wrongly provided by the organization. Our third case showed how the main developer favored a strongly agile approach, and used tools supporting that - while his organization was both incapable of understanding what he did things that way, and incapable of offering him support - as all their means were aimed at hierarchical methods. This, just like the above issues had an effect on his psychological needs - increasing his levels of frustration, and at times hopelessness, but perhaps more importantly it caused the development to be less efficient than it could have been - by preventing the developers from using their organization’s resources optimally.

A strong focus (either direct or indirect) on satisfying one orientor often lead to other orientors being paid little attention to, or being directly adversely affected. Effects from this seem to be especially noticable when a lack of satisfaction in an orientor is given a “quick and dirty” fix, an all too common approach in computing science. An
example of this would be in case 1, where the main actor personally ensured the needed budget, which did satisfy the existence orientor, but also directly caused a strong negative effect on his psychological needs, as the urge to succeed became (possibly unbearably) strong. Such short-term solutions thus seem to be little more than - indeed, patchwork that ignores the real underlying issue - here the fact that the organization was not invested enough in the need for the artifact that they were unwilling to provide a necessary budget. These relations between orients and how they influence each other also become apparent with the collapse of the development system in case 3, where the development system collapsed after the main actor was so fed up and quit, which because of a critical lack of actor variety meant there was simply no-one left to develop any further. This is essentially a reaction where unsatisfied psychological needs combined with an unsatisfied existence orientor caused the existence orientor to become strongly unsatisfied.

Most importantly, what we learn from this is that many reasons commonly given or accepted for the failure of development systems are often only symptoms of a deeper cause. Some of the above exemplifies this - where intuitively one would say that development collapsed because there was no-one to develop, the real underlying reason was the stress placed on the developers and the fact there were not enough of them - which in turn could be caused simply by an unrealistic plan for development, like the organization underestimating the needed finances and materials for development to succeed. That conclusion also leads us to another use of our method - that it often seems to be capable of reducing results of other methods that attempt to explain why development systems failed. The seven orientors seem to be indeed an exhaustive set, capable of either reducing confusing elaborate reasons of failure to the basic orientors, or more importantly, helping us to not place the blame for failure on the wrong thing. For instance, a commercial software risk methodology gives “changing requirements” as one major reason for failure. But, changing requirements are not the reason for failure here, the fact that the development system was incapable of dealing with them was. This should be understood in terms of the development system’s incapability to adapt itself to new insights and having too little freedom of action to deal with them, not by placing the blame solely on the fact that the requirements for the artifact changed over time. This does not mean our method should be used for every analysis, but it shows that it can be used in harmony with them to verify, and possibly refine their results.

A more problematic aspect of the method is the formalization and interpretation of its results. We wish to eventually have a large amount of data from which we can derive common sets of indicators, and from them create formal constraint sets to place on our model. By doing so we essentially make durability analysis a fairly simple instantiability problem - if we can populate our model with the case data it must be survivable. Formulating the constraints needed for this is no simple task, however. The difficulty in formalizing results of our method is two-fold: 
1. We derive negative and positive indicators - positive indicators being harder to formalize.
2. Formalizing the regulation of human behavior cannot be done solely on basis of logical possibilities.

The negative indicators we derive are fairly straightforward to formalize. If we see that in a case study some action is followed by negative effects, we can essentially make a rule prohibiting that behavior. But when we have a positive indicator - some action is followed by positive effects, it is not always correct or feasible to demand that action to always be taken. Furthermore, the issue with regulating human behavior is that we cannot merely create constraints that reason with logical possibilities. If we see that it is negative for some actor’s psychological needs that he is harassed by other actors, we can create constraints prohibiting that behavior - “Actors cannot harass other actors”, which is sadly logically impossible in any development system where actors can communicate with each other. It is however, deontically possible. Others also see the need for a distinction, as for instance the argument that most “Business Rules” are of a deontic nature and should be modeled as such (Halpin 2006).

For this reason we need to distinguish the constraints we make between those that are:
- logical possibilities, things that are necessarily so
- deontic possibilities, things that ought to be so

In order to demonstrate, we show the possibilities for creating a constraint based on the indicator “insignificant amount of end-users restrict possible amount and variety of necessary feedback”. We can formalize this by agreeing on some minimum number of end-users that would ensure the amount and variety of feedback, for instance 3. If we create a constraint that says “an Artifact is used by at least 3 End-users” we do not model what we want to ensure - namely that “an Artifact ought to be used by at least 3 End-users”. In ORM2 we can make this distinction by using a deontic constraint instead of an alethic one:

\[ \text{Alethic: is used by at least 3 \quad Deontic: ought to be used by at least 3} \]

With this distinction between types of constraints we both solve the problem of formalizing indicators that regulate human behavior, and positive indicators that do not always have to be true - as we can now formalize them as rules that ought to be obeyed, while understanding they can also be broken.

A remaining issue with the interpretation of our method’s results is the way in which we assign weights to each indicator. Because of a lack of empirical data from which to derive relative impacts of positive and negative events on specific development systems, we cannot yet implement these weightings - which might offer new subtle insights into the balance of orientor satisfaction in a case study. Practically this issue has to be resolved by means of empirical study, as there is little meaningful that can be done on solely theoretical basis. When such results are available, they would not be limited in usefulness to the method itself, but also offer an extra dimension to formalizing constraints, as we can then add more nuances for instance by adding different deontic expressions like “obligatory” and “permitted”, which would allow the model to be a more detailed abstraction of the real situation.

Finally, we must realize that when we deem a development system to be durable, that (3) it is but a temporal and local evaluation of its durability. There is absolutely no reason to assume, nor should we, that durability can be ensured once and then forgotten about. It is a property that needs a constant effort to be maintained (cf. Hjorth and Bagheri 2006), and as such we should constantly monitor our development systems if we want to constantly be aware of changes to its durability. This is a good thing, because as time goes on, we can become aware of
new factors influencing the durability of our development systems, integrate them into our understanding and become more and more adept at ensuring the durability of our development systems.

Furthermore, while any analysis will always be of limited scope, we need to understand that the environment of our development systems is equally as important, both for its own durability, and for the effects it has on the durability of systems in that environment. We should not think only of ensuring our own durability and create artifacts without regard for how they will interact with others. Such situations have occurred many times before, not only in computing science, and time after time prove to be the undoing of the artifacts those systems created, as exemplified by the development of the “Ada” programming language. Conceived as an all-purpose, all-encompassing new language that would supersede all other languages, its developers either forgot or did not care for the problems that would arise when all existing software would have to be rewritten to this language - a feat needing significant amount of times for even the smallest programs as was shown by (cf. Huisingan et al. 1986).

Conclusions

This thesis presented a fundamental view on system development in computing science, and adopted a method for sustainable development as a means to measure durability of development systems. This method was applied in a small sample of case studies to derive durability data, and found by the author to be a simple and effective way of estimating the durability of a given development system.

There are several possible directions to go in future research: (1) Using the method to analyze development methods and paradigms instead specific cases. As the method can easily be adapted to this use, we could investigate whether specific methods place more undue stress on the satisfaction of basic orientors than others, and from that attempt to advice which environments are best suited for which methods of development (i.e. an environment with focus on adaptability would do wisely to avoid methods that are inherently not adaptable, like the agile versus hierarchical development argument) Or, (2), build further on the method and view shown in this thesis, by following the following steps:

- Create more a more detailed model (of a general) development system, so as to be able to model an as large amount of cases as possible.
- Undertake an exhaustive, retrospective case study, aiming to:
  - Find significant common occurrences of indicators, and generalizing them
  - Find relative contextual impacts (i.e. weights) of those indicators
- Construct a formalized constraint set, based on the above, which can be placed on the created model. This can be verified by resolving a significant amount of cases by checking their instantiability.
- Finally, use the new model and constraint set to model current and ongoing developments, both attempting to predict their durability, point out weak points, and, if in co-occurrence with future research (1), attempt to advise whether the used development method is a good fit

Further proposed development of the method

Finally, with the data resulting from the above developments a (5) system dynamics model to analyze the effect of individual orientor’s satisfaction on each other could be created. This would aid in the understanding of how much attention could be given to an orientor’s satisfaction before it has negative effects elsewhere - or whether explicit attention could be given at all without negatively affecting another orientor.

Acknowledgements

Many thanks to Hanno Wupper, Stijiu Hoppenbrouwers, Luca Consoli, Marinus van Herpen and Alina Vladimirovna Lartseva for much inspiration, critical discussions and useful anecdotes.

Bibliography

Annotations are provided that may be helpful to the reader wanting more information about specific topics or concepts that this work based itself on, or was inspired by.


The author offers an argument as to why the concept of sustainable development is not strictly necessary and most interpretations of it are already accounted for by other leading theories.


This report includes a major portion of Bossel’s work and ideas on indicator-driven sustainable development analysis, and as such is a useful resource for a background and concepts needed for a systematic approach to the problem of sustainability.


This book includes much research on direct and indirect forms of speech, most importantly here teaching us accepted differences between them, corroborating our understanding from (Pruitt 1971), thereby making our communicative view of the second-order information system and the importance of both direct and indirect speech in it grounded in a wealth of existing research.


The author provides an argument that computing science is inherently about discovering


The authors propose a method of judging sustainability of (for example agricultural) systems based on three specific dimensions: environmental, social and economic. While of different focus, their approach shares the idea of using a limited amount of specific aspects according to which a system is judged. However, the difference of (1996) method is the more open approach by adopting three dimensions which are much more open to interpretation.


The author proposes that studies of human development should be used to better system development, by making such development systems more “human-environment friendly”. While the method and theoretical background based on teachings from the al-Qur’an are both a familiar background, nor one that (because of political and cultural factors) can expect a warm welcome in Western studies of system development, it is significant that in more areas of study the human being is put central to system development.


This paper disseminates the (then) different understandings of “prototyping” as it related to systems development, and offer arguments how it fits the real world situation more aptly than the old systems development paradigm. Interesting for us is how this argument continues to be (correctly) used to make sure our development methods are suited to the real world around us - culminating in more agile, perhaps chaotic methods to deal with developing artifacts in the chaotic world around us.


While much more influential and larger in scope, what we can learn from Weizenbaum here is that the human being is, and even if ever possible should not be replaced as the central point of any development system, and therefore should be studied as being that central point. This concept, and the understanding of the difference between reasoning and deciding, is a vital concept that should be understood and conveyed when attempting to reason about the second-order information system.


The authors provide a well-constructed overview not only of what computing science is, but more importantly provide clear and concise definitions of some of the important terminology we (mis)use all too often.


Appendix

The following information is given for the sake of reproducibility and those who wish to use parts of our work for their own goals. We give the ways in which we gathered our information (by means of the interview structures), processed it (by means of the numerical case study data) and structured our information (by means of the model verbalization).

Structure of held interviews

Interviews were kept open as to gather as much information as possible by not constricting what the interviewed would talk about. The main questions asked were:

- What was the reason you needed software? What was the real thing, ability that you were missing?
- How did you begin? Was there a backing from your organization, or did you more simply start and planned to create a basis as you went along?
- Why did you decide to create something new, or implement something existing?
- What were some significant events, according to you, that happened during the development, that had an effect on your perceived chance of the project reaching its goals?
- What did you do, or did your organization do when such goals were reached, reached too late, or not reached at all?

When the interviewed answered these main questions, further, less structured questions were asked concerning the basic orienters, like:

- Were there at any time, perhaps because of decisions you took, any problems with the numbers of people working on your project, or the budget you had? (existence)
- Did you have a target audience when you (first) delivered your products? And did you manage to interact with them to refine your ideas of what they wanted? (effectiveness)
- And when those goals seemed unreachable, did you attempt to approach them by other means? Was your organization supportive of that? (freedom of action)
- What steps did you take to prevent, or deal with your project from stalling when vital persons or assets were absent? (security)
Were you willing to change, or modify your goals, your milestones or methods to better suit your milestones and what your target audience wanted? And were you able to do so? (adaptability)

Did you have any cooperation with other organizations, or departments, to help you out on aspects you were less familiar with yourself, or simply could not spend much time on? (coexistence)

Did you at any time feel particularly stressed out at having to make certain milestones, or felt like the organization was not appreciable enough towards you, or made it unneeded difficult to continue development? (psychological needs)

Case study data

The case study data was derived as a chronological list of (key) events that occurred during development. They are shown here first in raw layout, being a simple listing of which orientor was affected in which subsystem to which polarity. This is followed by that data processed with a weighting of positive = 1 and negative = -1, giving scores for all subsystem/orientor combinations in the same structure as shown in Table 1. From these net scores were calculated both vertically (for each subsystem), and laterally (for each orientor), which were used to generate the radar charts included in the case study.

Raw

Case 1
co-existence, organization, positive
security, human, negative
co-existence, human, positive
existence, organization, negative
security, human, negative
psychological needs, human, negative
effectiveness, human, negative
effectiveness, human, negative
psychological needs, human, negative
adaptability, human, positive
freedom of action, organization, positive
co-existence, technology, positive

Case 2
effectiveness, human, positive
adaptability, human, positive
co-existence, technology, positive
co-existence, human, positive
existence, technology, positive
effectiveness, human, positive
freedom of action, organization, positive
freedom of action, human, positive

Case 3
security, human, negative
co-existence, technology, negative
existence, organization, negative
security, technology, negative
existence, organization, negative
effectiveness, human, positive
adaptability, organization, negative
effectiveness, human, negative
freedom of action, human, negative
psychological needs, human, negative
adaptability, human, negative
existence, human, negative
co-existence, technology, positive

Processed

Case 1

<table>
<thead>
<tr>
<th>orientor/subsystem</th>
<th>human</th>
<th>tech.</th>
<th>org.</th>
<th>net score</th>
</tr>
</thead>
<tbody>
<tr>
<td>existence</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>effectiveness</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>freedom of action</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>security</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>adaptability</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>co-existence</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>psychological needs</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>net score</td>
<td>-3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Case 2

<table>
<thead>
<tr>
<th>orientor/subsystem</th>
<th>human</th>
<th>tech.</th>
<th>org.</th>
<th>net score</th>
</tr>
</thead>
<tbody>
<tr>
<td>existence</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>effectiveness</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>freedom of action</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>security</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>adaptability</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>co-existence</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>psychological needs</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>net score</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Case 1

<table>
<thead>
<tr>
<th>orientor/subsystem</th>
<th>human</th>
<th>tech.</th>
<th>org.</th>
<th>net score</th>
</tr>
</thead>
<tbody>
<tr>
<td>existence</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>effectiveness</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>freedom of action</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>security</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>adaptability</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>co-existence</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>psychological needs</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>net score</td>
<td>-5</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
</tr>
</tbody>
</table>

Model verbalization

This is the verbalization of our model of a general development system. It is given in order to more easily make it possible to reconstruct the model in NORMA or similar software for those interested, and for the sake of future use as the framework in which case studies should be instantiated.

Text is an entity type.
Reference Scheme: Text has Text.id.
Reference Mode: .id.
Fact Types:
Text has Text.id.
Statement makes up Text.
Text is TextType.
Property is fulfilled by Text.
Text makes up Blueprint.

Statement makes up Text.
It is possible that more than one Statement makes up the same Text and that more than one Text consists of the same Statement.
In each population of Statement makes up Text, each Statement combination occurs at most once.
This association with Statement, Text provides the preferred identification scheme for StatementMakesUpText.
Each Text consists of some Statement.

Statement has consensus.
Reason is a value type.
Portable data type: Text: Variable Length.

Fact Types:
Statement was rejected with Reason.
Statement was rejected with Reason.
It is possible that more than one Statement was rejected with the same Reason and that the same Statement was rejected with more than one Reason. In each population of Statement was rejected with Reason, each Reason Statement combination occurs at most once. This association with Reason, Statement provides the preferred identification scheme for StatementWasRejectedWithReason.

TextType is a value type. Portable data type: Text: Variable Length.

Fact Types: Text is TextType. The possible values of TextType are 'Code', 'Specification', 'Document'. Text is TextType. Each Text is exactly one TextType. It is possible that more than one Text is the same TextType.

Actor is an entity type. Reference Scheme: Actor has Actor_name. Reference Mode: .name.

Fact Types: Actor has Actor_name. Statement made by Actor (.id). Actor is ActorType. Actor is from Organization. Actor states needed Property. Actor makes Statement.

ActorType is a value type. Portable data type: Text: Variable Length.

Fact Types: Actor is ActorType. The possible values of ActorType are 'Coder', 'Specifier', 'Expert', 'Contractor'. Actor is ActorType. It is possible that more than one Actor is the same ActorType and that the same Actor is more than one ActorType. In each population of Actor is ActorType, each Actor, ActorType combination occurs at most once. This association with Actor, ActorType provides the preferred identification scheme for ActorIsActorType. Each Actor is some ActorType.

Statement conflicts with Statement. It is possible that more than one Statement conflicts with the same Statement and that the same Statement conflicts with more than one Statement. In each population of Statement conflicts with Statement, each Statement, Statement combination occurs at most once. This association with Statement, Statement provides the preferred identification scheme for StatementConflictsWithStatement.

Organization is an entity type. Reference Scheme: Organization has Organization_name. Reference Mode: .name.

Fact Types: Organization has Organization_name. Artifact was implemented in Organization.

Actor is from Organization. It is possible that more than one Actor is from the same Organization and that the same Actor is from more than one Organization. In each population of Actor is from Organization, each Organization, Actor combination occurs at most once. This association with Organization, Actor provides the preferred identification scheme for ActorIsFromOrganization. Each Actor is from some Organization.

Property is an entity type. Reference Scheme: Property has Property_id. Reference Mode: .id.

Fact Types: Actor states needed Property. Property is fulfilled by Text. Property make up Specification. Property has Property_id.

Actor states needed Property. It is possible that more than one Actor states needed the same Property and that the same Actor states needed more than one Property. In each population of Actor states needed Property, each Property, Actor combination occurs at most once. This association with Property, Actor provides the preferred identification scheme for Need. For each Property, some Actor states needed that Property.

Date is a value type. Portable data type: Temporal: Date.

Fact Types: Need is stated for Date. Implementation was done on Date. Notes: A statement must conflict with another statement.

Need is stated for Date. Each Need is stated for at most one Date. It is possible that more than one Need is stated for the same Date.

Property is fulfilled by Text. It is possible that more than one Property is fulfilled by the same Text and that the same Property is fulfilled by more than one Text. In each population of Property is fulfilled by Text, each Property, Text combination occurs at most once. This association with Property, Text provides the preferred identification scheme for PropertyIsFulfilledByText.

Blueprint is an entity type. Reference Scheme: Blueprint has Blueprint_id. Reference Mode: .id.

Fact Types: Text makes up Blueprint. Specification is realized by a Blueprint. Blueprint has Blueprint_id.

Text makes up Blueprint. It is possible that more than one Text makes up the same Blueprint and that the same Text makes up more than one Blueprint. In each population of Text makes up Blueprint, each Text, Blueprint combination occurs at most once. This association with Text, Blueprint provides the preferred identification scheme for TextMakesUpBlueprint. For each Blueprint, some Text makes up that Blueprint.


Property make up Specification.
It is possible that more than one Property make up the same Specification
and that the same Property make up more than one Specification.
In each population of Property make up Specification, each Specification
, Property combination occurs at most once.
This association with Specification, Property provides the preferred identification
scheme for PropertyMakeUpSpecification.
For each Specification
, some Property make up that Specification.
Specification is realized by a Blueprint.
It is possible that more than one Specification is realized by a the same Blueprint
and that the same Specification is realized by a more than one Blueprint.
In each population of Specification is realized by a Blueprint, each Blueprint, Specification
combination occurs at most once.
This association with Blueprint, Specification provides the preferred identification scheme for Artifact.
Artifact was implemented in Organization.
It is possible that more than one Artifact was implemented in the same Organization
and that the same Artifact was implemented in more than one Organization.
In each population of Artifact was implemented in Organization, each Organization
, Artifact combination occurs at most once.
This association with Organization, Artifact provides the preferred identification scheme for Implementation.
Implementation was done on Date.
Each Implementation was done on exactly one Date.
It is possible that more than one Implementation was done on the same Date.
Environment is a value type.
Portable data type: Text: Variable Length.
Fact Types:
Artifact is influenced by Environment.
Artifact is influenced by Environment.
It is possible that more than one Artifact is influenced by the same Environment
and that the same Artifact is influenced by more than one Environment.
In each population of Artifact is influenced by Environment, each Artifact
Environment combination occurs at most once.
This association with Artifact, Environment provides the preferred identification
scheme for ArtifactIsInfluencedByEnvironment.
Statement is an entity type.
Reference Scheme: Statement has Statement.id.
Reference Mode: .id.
Fact Types:
Statement has Statement.id.
Statement makes up Text.
Statement has consensus.
Statement was rejected with Reason.
Statement made by Actor (.id).
Statement conflicts with Statement.
Actor makes Statement.
End-user is a value type.
Portable data type: Text: Variable Length.
Fact Types:
Artifact is used by End-user.
Artifact is used by End-user.
It is possible that more than one Artifact is used by the same End-user
and that the same Artifact is used by more than one End-user.
In each population of Artifact is used by End-user, each Artifact
End-user combination occurs at most once.