How Computer Programmers Work

Understanding Software Development in Practice

Chapter 2

Programming practice

2.1 *Tribeflame*

In order to learn something about programming, we need to take a look at a programming process. In Chapter 3 (Programming theories), we will study the models of programming processes that are already found in the literature. But any model of programming processes must necessarily be an idealisation, to some degree; and while we are interested in finding out the ways in which these idealisations are useful, we cannot begin with the models. We have to start from the concrete, real work processes that the models represent.

Therefore in this chapter, we will study the work process of the company Tribeflame in Turku, South-western Finland, during four weeks in August and September 2011.[1] Tribeflame is a small company

[1] I spent a total of 17 workdays in the period 15th August – 9th September 2011 observing the work at Tribeflame. I would normally be in the office with a notebook whenever the developers were present. Occasionally I talked to them about their work. I also interviewed all of the developers formally, and had two of the

Figure 2.1 – The office of Tribeflame with some of the developers' desks. September 2011.

that makes computer games for tablet computers, primarily Apple's iPad. At the time of the study period, Tribeflame had completed six games. Over the course of the study, the company worked on developing three different games, though most of the time was spent on one of these. At the time, Tribeflame consisted of the two founders, two programmers, and two graphic artists, though out of the six one worked part time, and one was only employed for the summer.

Is the development process used by Tribeflame typical? There is

programmers explain their source code to me. See the list of source material, page 245. The interested reader can find more information about my observation method as well as an explanation of its theoretical background in an article I wrote during my research – see Suenson 2013.

no reason to doubt that it is a typical small game company.[1] However, we are not really interested in the question of whether Tribeflame's process is typical of programming in general: any real process will have its own peculiarities and in this sense be atypical. We are looking for some cultural traits and constraints on the possible forms computer programming can take, and for this purpose Tribeflame serves very well as an example. In Chapter 5 (Safety critical programming), we will take a look at some very different development processes.

2.2 *The development team*

Björn and Andreas[2] are the founders and owners of Tribeflame. They became friends in university and started Tribeflame together after having worked as employees for other companies for five or six years after graduation. Björn acts as the company's Chief Executive Officer (CEO), and takes care of most of the administrative side of running the business. The largest part of his work, however, is to do the game level design for Tribeflame's games. That means thinking up the puzzles that the player has to solve when playing the games. Björn has the final say regarding how the games look and work. Andreas acts as the Chief Technological Officer (CTO): he works mainly as a programmer, and has the final say in technical decisions.

Mickie is employed full time as a programmer. Before Tribeflame, he worked as an employee elsewhere in the IT industry. Björn, Andreas, and Mickie all have technical IT degrees from the same Finnish

[1] In 2010, 4473 out of 4981 Finnish software companies had 9 or fewer people working in them. In 2011, 3% of Finnish software companies (around 150) were in the game industry. Rönkkö & Peltonen 2012.

[2] The names of the owners and employees of Tribeflame have been changed to protect their privacy. The name of the company is the real name.

university.[1]

Matti is employed as a full time graphic artist in Tribeflame, responsible for producing almost all the graphics for the games. Fredrik, a programmer, is finishing his education at the university while working part time; he is about to start his own company in an unrelated business, which means that he is slowly ending his time with Tribeflame. Kati is a graphic artist who was employed for the summer to replace Matti while he was on vacation.

Tribeflame mostly works as a team of individuals with specialized competences. The graphic artists Matti and Kati do not have the skills to do what the programmers, Andreas, Mickie, and Fredrik, are doing, and vice versa. Decisions about running the business are taken jointly by Björn and Andreas and do not involve the employees. Besides the administrative decisions, Björn and Andreas share the formal decision competence, with Björn being in charge of product decisions and Andreas in charge of strategic technical decisions. However, most of the decisions regarding the products (the games) are arrived at in a common process of discussion and decision that involves everyone in the company.

2.3 *The rhythm of work*

The work at Tribeflame has a certain rhythm. Most of the work takes place in the company's office, a single room in an office building next to the university's IT facility in Turku (see Figure 2.1). At times, everyone is focused on their own tasks: the room is silent except for the clicking of mice and the tapping of keyboards, and the concentration

[1] Åbo Akademi, the Swedish language university in Finland.

is almost palpable.[1] At other times the room is alive with laughter and jokes – people constantly interrupt each other and show each other games and graphics on their screens. On frequent occasions, everyone leaves their desks and gathers together around the table in the centre of the room in order to have a meeting.

In Figure 2.2 we see the three main activities that are usually carried out alone, and how often they happen. Coding refers to the part of programming that is actually entering the programming code on the computer, as opposed to planning and talking about it. This task is done by the programmers. Graphics and animation refers to drawing the pictures and animations used in the games, and this is done by the graphic artists. Level design refers to thinking up the puzzles the player has to solve and entering them into the game in a suitable form. This task is done by Björn, the CEO.

We can see that these tasks are very frequent. Coding and graphics work happens every day; game level design happens whenever Björn is free from more important duties, though still quite frequently. These are the tasks that directly contribute to the games, so it is not surprising that they occur so often.

However, equally important as the time spent working alone is the interaction within the company. Figure 2.3 shows the occurrence of team meetings, which happen almost as frequently as the activities that contribute directly to the games. Occasionally the meetings are short but commonly last for an hour or two. On Fridays the meetings sometimes take up most of the day. During the final two weeks of the observation period, Björn instituted a daily meeting. Note that

[1] Robert Willim describes a similar atmosphere in the Swedish IT company Framfab: "As mentioned, a quiet sense of being busy prevailed in the Ideon office. The mood of relative ease was partly due to much of the awareness being focused on the interface to the technology. The employees' concentration and focus was on what Steven Johnson (1997) calls the data rhythm. From this arises a contemplative concentration, trained inwards and at the same time connected to technology." Willim 2002 p. 84.

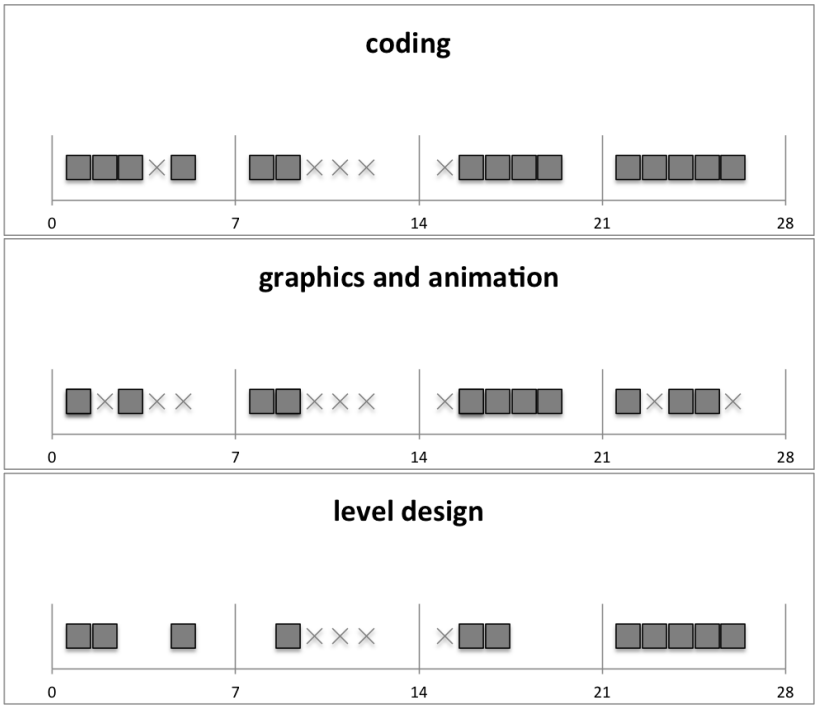

Figure 2.2 – Frequency of tasks at Tribeflame during the observation period. The axis represents days. The boxes indicate that the activity occurred on that day, the crosses indicate missing observation data.

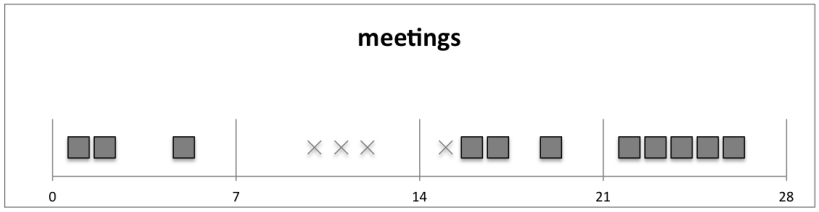

Figure 2.3 – Frequency of team meetings within Tribeflame. The axis represents days. The boxes indicate that the activity occurred on that day; the crosses indicate missing observation data.

Figure 2.3 shows only internal meetings in the company – Björn and Andreas also have meetings with external contacts.

This means that the work within Tribeflame is forever rhythmically shifting between working more or less individually on tasks, and coming together to discuss the tasks. If viewed from above, the work in Tribeflame looks like the schematic drawing in Figure 2.4: the developers move from their desks at the periphery of the room to the meeting table in the centre; then they move back away from the centre to the periphery, and so on and so forth.

The meetings at the central table are not the only form of exchange between the developers. As shown in Figure 2.5, interaction is just as frequent in the form of discussions, informal chat, banter, evaluation of each others' work, or working together on solving tasks. Schematically, the drawings of Figure 2.4 have to be complemented by the one in Figure 2.6, which shows the interaction that takes place between the developers when they are not in meetings.

The time spent on interaction is significant. As an example, Figure 2.7 shows the work interaction of Andreas on an ordinary Wednesday.[1]

[1]This particular day was chosen as an example in an attempt to find a typical work day: although it should be noted that all the observed days had some atypical features.

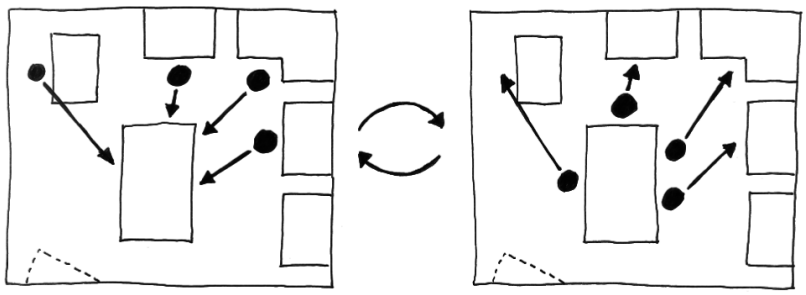

Figure 2.4 – The work at Tribeflame shifts from the individual desks in the periphery of the room to the meeting table in the centre (left); then it shifts back again to the periphery (right). This process goes on continually.

On this day, everyone spent more than half their time working with others. Over a third of the time was spent with everyone in a meeting around the central table (four people were at work during this week). Around a fifth of the time was spent in interactions in smaller groups, discussing and chatting.

As we see from Figure 2.7, there are two main blocks of time spent alone: one in the morning and one after lunch, beginning at around 220 minutes. Each of these blocks are followed by a meeting period involving everyone. Other types of interaction are spread throughout the day. Thus we see both the rhythmic movement between centre and periphery, which is illustrated in Figure 2.4, and the individual interactions shown in Figure 2.6.

The lesson of this is that the Tribeflame work process is not only a coordinated effort between specialized individuals – it is also an intensely social and communicative process in which common discussions are central. The function of the discussions is to build understanding in the company. Decisions are then made on the basis of this common understanding, which is constantly evolving and corrected

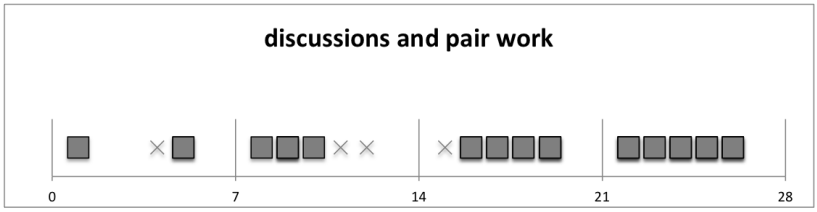

Figure 2.5 – Frequency of discussions, work-related chats, and working together on tasks. The axis represents days. The boxes indicate that the activity occurred on that day; the crosses indicate missing observation data.

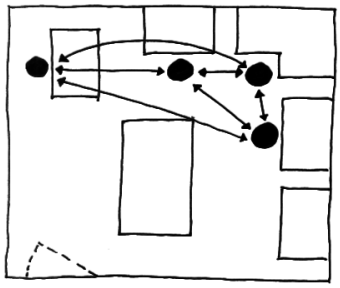

Figure 2.6 – The developers also interact frequently outside of meetings.

by the developers. Though the developers have specialized roles, they all contribute to the common direction of the company's games, and it is important that they are all heard. For example, though Kati is a temporary employee she is expected, even on her last work day, to participate in the development meeting and contribute her opinions,[1] regardless of the fact that she will not be involved in the development process anymore.

The focus on common understanding and consensus-building discussions means that an authoritative decision is rarely made. For ex-

[1] Field diary, Friday September 2nd 2011, 11:00–12:24.

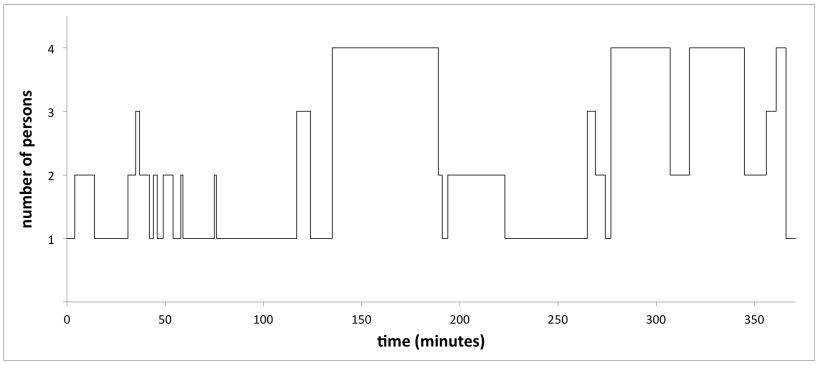

Figure 2.7 – Work interaction for Andreas, Wednesday 7th September 2011, 9:00–15:00. When the number of persons is one it means that he is working alone; when it is greater than one that he is working together with others. The number of persons at work this week was four, so the high points in the graph indicate the whole company working together.

ample, after a lengthy discussion between the developers about where on the screen the game menu should be placed, Björn finally takes a decision because agreement cannot be reached: "In the end I decide that it will be moved to the right. Sorry guys."[1] Though Björn clearly has the authority to make the decision, and it is socially acceptable for him to do so, the fact that he feels the need to give an apology shows that this is not the normal way of reaching a decision in the company.

The focus on understanding also means that the kind of communication that could be called "information sharing" is relatively infrequent: if we understand information sharing to be a process whereby one person gives some information to another person, who mainly listens and can ask clarifying questions. As can be seen in Figure 2.8, information sharing is much less frequent than the more collabora-

[1] Field diary, Thursday 8th September 2011, 16:15.

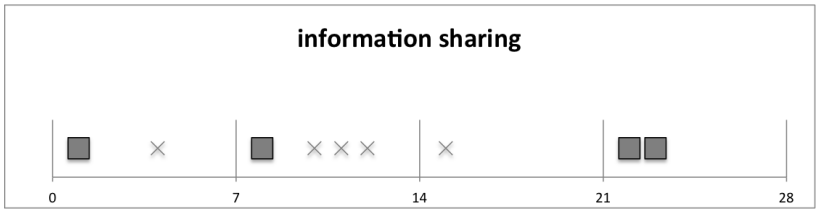

Figure 2.8 – The relative infrequency of information sharing. Information sharing is when one person provides the information and the other merely listens and can ask clarifying questions. The axis represents days. The boxes indicate that the activity occurred on that day, the crosses indicate missing observation data.

tive meetings and discussions shown in Figure 2.3 and 2.5. That the communication patterns in Tribeflame are characterized more by discussions than by information sharing is somewhat paralleled by the decision process, which can better be described as "decision reaching" than as "decision making". The term "decision reaching" emphasizes that agreement and exchange plays a much larger role than in a process where decisions are simply made on the basis of the best available information.

2.4 *Playing and planning*

Playing the games that are being developed is an important part of the work process. For this, I use the term "playtest". The developers continually run the game to see how some detail has turned out, but this is not what is meant by playtest. Rather, it is playing the game for a longer period to see how it works as a whole and how much fun it is.

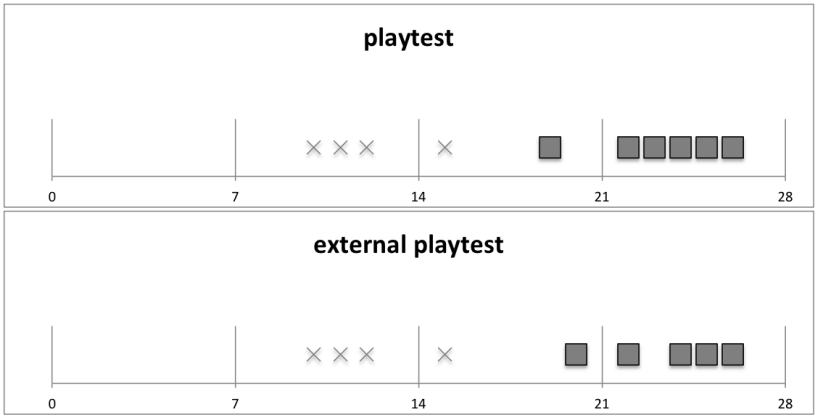

Figure 2.9 – Frequency of playtest by the developers themselves (top), and by persons outside Tribeflame (bottom). The axis represents days. The boxes indicate that the activity occurred on that day; the crosses indicate missing observation data.

In this sense, there are two kinds of playtest: one when the developers themselves play their game, and another ("external playtest") when they get someone from outside the company to play it and talk about their experience.

Figure 2.9 shows the frequency of internal and external playtest. It indicates that the developers do not start to perform playtests until near to the final week of the observation period. At this time, however, playtests become a nearly daily occurrence, and the tests have a large influence on the discussion topics and meetings in the company. For the external tests, the developers ask colleagues in the game industry, colleagues in other branches of industry, relatives, spouses, and even me – almost everyone with whom they come into contact and can persuade to try the game.

Besides playing their own games, the developers also frequently

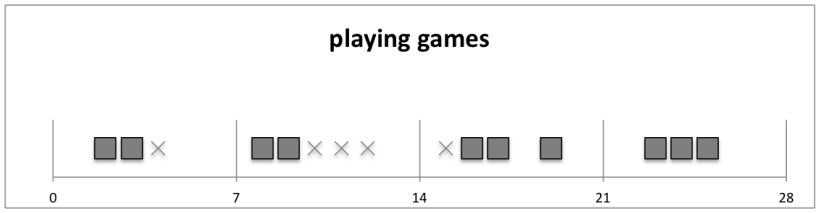

Figure 2.10 – Frequency of playing other companies' games during work hours. The axis represents days. The boxes indicate that the activity occurred on that day; the crosses indicate missing observation data.

play other companies' games and talk about them. Figure 2.10 shows the frequency of playing other games during working hours. Since the company was only observed during work hours, there is no data on how often other companies' games are played outside work hours; but this clearly happens as it frequently features as a topic of conversation.

Planning in Tribeflame is neither very systematic nor very long term. There is a general idea about where the game is headed, but this is rarely written down or captured in a form other than concept sketches or similar. Tasks are usually planned one week ahead at the Friday meeting; they are written on a flip-chart or whiteboard and crossed out during the week as they are completed. New tasks that are discovered during the week are added to the whiteboard immediately. The tasks are frequently discussed during the course of the work. On an individual level, the developers sometimes write "to-do lists" for themselves on a piece of paper.

A humorous illustration of the *ad hoc* approach to planning in Tribeflame comes when Björn has difficulty wiping the whiteboard clean of old scribblings before a meeting. Instead of going in search of some spirit meant for cleaning he uses a small bottle of Minttu (Finnish mint liqueur) that happens to be standing around in the

office, exclaiming: "whatever".[1] Though the gesture does not strictly have anything to do with the planning process, it illustrates very well the mindset in Tribeflame towards planning: do whatever works.

From a business perspective, the goal of Tribeflame is to produce games that can be sold and generate enough profit to sustain the development. This, however, does not explain what it is that makes it possible to sell a game, or how to make one. Mickie explains that "the code is just a means to reach a goal".[2] The game customers do not care about Tribeflame's business goal: they have their own goal, which is to have fun.

Consequently, many of the discussions at Tribeflame revolve around whether the games are fun and what it means for a game to be fun – both their own games and those developed by other companies. The developers are conscious that what they are doing is a form of entertainment. When giving a presentation about the company, Björn says that "entertainment is the constant in the company."[3] Andreas justifies charging money for the games by comparing them to other forms of entertainment: "it's a form of entertainment, it's fair to pay for games. People pay ten euro for two beers at a café"[4] – meaning that the entertainment value justifies charging more than the production costs.

[1] Field diary, Friday 2nd September 2011, 11:11.
[2] Field diary, Thursday 1st September 2011, 13:00.
[3] Presentation of Tribeflame at Åbo Akademi, Tuesday 4th December 2011.
[4] Field diary, Wednesday 7th September 2011, 13:50.

Chapter 4

Game programming

4.1 *Analysis with programming theories*

In Chapter 2 (Programming practice), we discussed the work process of Tribeflame, a small computer game company of six people in Turku, South-western Finland. In Chapter 3 (Programming theories) the most important theories of programming were explained. In this section, we will apply the mainstream theories of programming to Tribeflame's process in order to see how well they are suited to analysis of a concrete programming process. The purpose of this is not to indicate where the mainstream theories are wrong, but to investigate whether they work as general programming theories. If they are truly general programming theories, they should be able to explain a viable commercial process as provided by the Tribeflame example. If they are not, science demands that we explain to what situations the theories are suited, and what their limitations are.

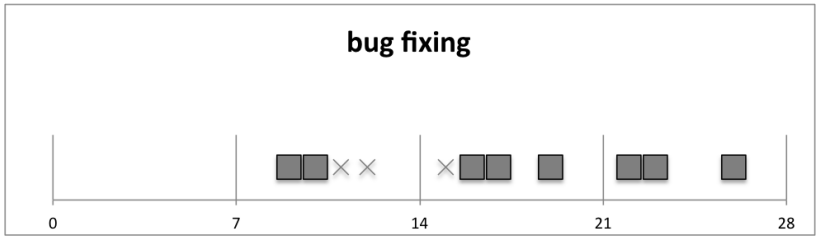

Figure 4.1 – Frequency of bug fixing, a part of testing. The axis represents days. The boxes indicate that the activity occurred on that day; the crosses indicate missing observation data.

4.1.1 *Software engineering*

Looking at the Tribeflame process, we see a distinct lack of the clearly-separated phases that are the basis of software engineering theories. The phases that are common to nearly all theories are specification, architecture (or high level design), design, coding, and test. Of these, the specification, architecture, and design activities at Tribeflame are carried out during the meetings, shown in Figure 2.3 (page 37), and during the constant discussions and interactions outside meetings, shown in Figure 2.5 (page 39). As indicated, the activities are spread out over the duration of the process.

The same is the case with coding, as can be seen in Figure 2.2 (page 36). Testing comprises several activities. According to traditional software engineering, bug fixing (error fixing) can be considered a part of testing. Bug fixing at Tribeflame is shown in Figure 4.1. Playtests and external playtests, shown in Figure 2.9 (page 42), also function as forms of testing. As we can see, testing occurs throughout the observation period except for the first week, and testing is concurrent with the specification, design and coding activities.

Even though the observation period covers but a small fraction of the development of a complete product, we find the activities of all of the phases of traditional software engineering represented within it. Moreover, the activities do not occur in an orderly, separated fashion, so we cannot interpret the development process as consisting of smaller iterations, each containing a complete set of phases, as advocated by some software engineering theories.

The conclusion is that the activities described by the software engineering phases all occur in Tribeflame's process, but the idea of separating the activities into distinct phases does not help to explain what is happening in the process.

Corresponding to the lack of separate phases is a lack of transition criteria. In software engineering, transition criteria between phases are normally the completion of a document, work product, or artefact. In Tribeflame's work, there are remarkably few documents, and they do not play a central role. A few concept sketches and similar are important, and these are lying around in the office. But they are not kept up to date, and they are seldom referred to.

Most of the knowledge is transmitted orally or in the game itself. The source code of the game can conceivably be thought of as a work product, but it does not function as a transition point since it is constantly evolving and never reaches a finished state that is signed off.

As we saw in Section 3.1 (Software engineering), the main concerns of software engineering are planning, documenting, and evaluating. We can recall from Chapter 2 (Programming practice) that in Tribeflame planning is done in an impromptu way on temporary flip-charts and whiteboards – far from the systematic planning and milestone-oriented methods promoted by software engineering, such as PERT and Gantt charts.

Regarding documentation, Tribeflame produces almost none. The knowledge is kept in the developers' memories, in the evolving source

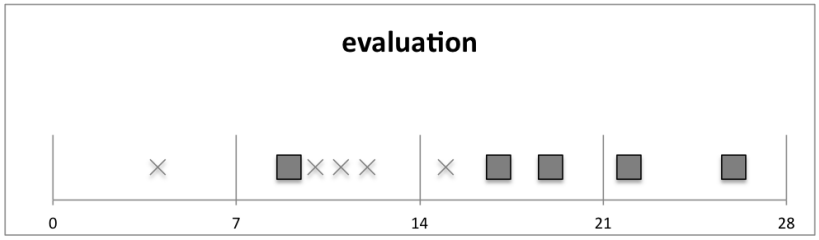

Figure 4.2 – Frequency of evaluation sessions. The axis represents days. The boxes indicate that the activity occurred on that day; the crosses indicate missing observation data.

code, and occasionally concretized in a few concept sketches.

Whereas planning (in a software engineering sense) and documentation are not major aspects of Tribeflame's work, evaluation is prominent. This is often a significant feature of meetings at Tribeflame, as indicated in Figure 2.3 (page 37). In addition, distinct evaluation sessions sometimes take place, shown in Figure 4.2. Thus, of the three major concerns of software engineering, evaluating seems to be the concept that is most useful for describing what is going on during the process observed, but it only applies to a part of Tribeflame's work.

As we saw in Figure 2.10 (page 43), playing other companies' games during work hours is a frequently-occurring activity. It is hard to characterize this important activity within the conceptual framework of software engineering, as it clearly does not fit into any of the categories of architecture, design, coding, or testing. It could perhaps be perceived as a kind of research necessary for requirements specification. However, specification is supposed to be as complete as possible before starting the design work, and this does not fit well with the ongoing activity of game playing.

Some might say that it is unfair to try to analyse Tribeflame's process with software engineering terms, as software engineering is specifi-

cally oriented toward large projects, and not small teams. Software engineering theory is useful in the right circumstances. In Chapter 5 (Safety critical programming) we will consider examples of software engineering theory being put to good use in practice. Nevertheless, as we have seen above, traditional software engineering theory is inadequate for explaining this example of a small team development process. Since 90 per cent of Finnish software companies are of a similar small size, this is a real challenge to the explanatory power of software engineering.[1]

From a research perspective, it is not necessarily a problem that software engineering theory might only be applicable to a minority of companies within the software industry.[2] From a programming perspective, however, this can have serious consequences. Most programmers learn about software engineering during their education, but not so much about the limitations of software engineering. When presented with a real software process, many will unconsciously perceive it in terms of the software engineering theories they have learned, even when the process is not suited to software engineering analysis, as is the case in Tribeflame's example. Not being aware of the limitations of software engineering theory then has the unfortunate effect of making the programmer apply the wrong mental tool to his task.

4.1.2 *Agile development*

The diversity of Agile approaches makes it rather challenging to use Agile concepts for analysing a concrete process, for it is not given

[1] In 2010, 90% of Finnish software companies had 9 people or fewer. Rönkkö & Peltonen 2012. See note 1 on page 33.

[2] Though it is of course a serious problem if the limitations of the theory are not clearly understood and presented.

which of the Agile approaches should be used. It is immediately clear, however, that on a practical level, Tribeflame's process corresponds only sporadically to Agile concepts.

To take one example: Tribeflame's work corresponds to the Extreme Programming practice of placing all developers together in the same room, simply because Tribeflame has only one office room available. But there is no correspondence to the practice that all program code should be tested every day, as Tribeflame does not have an automated testing system in place, much less test cases for all of the code. This, of course, stems from the nature of their product – it is nearly impossible to design test cases that determine whether an interactive computer game is "correct". Likewise, the practice of programmers coding together in pairs is not followed, as Tribeflame's developers spend much more time working alone than in pairs. There are more practices in Extreme Programming, but it is clear that Tribeflame's process does not resemble Extreme Programming to a significant extent.[1]

In another example, we can look at how well Tribeflame's work corresponds to the Scrum rules. The three important roles in Scrum are the Product Owner, the ScrumMaster, and the Team. Tribeflame has neither Product Owner nor ScrumMaster. The Team, understood as everyone working in Tribeflame, is of course very important in Tribeflame; but it is not exactly the same as the Scrum conception of a Team. In Scrum, the Team has to be without internal hierarchy; in Tribeflame, Björn and Andreas, as the owners, have the formal decision-making authority, even if they seldom exercise their authority directly.

Those working at Tribeflame do not use the Scrum version of iterations (Sprints); they seldom explicitly prioritize their tasks, and they

[1]There are 13 primary practices and 11 corollary practices in Extreme Programming. Beck & Andres 2004.

do not commit to a specific workload each month. Sprint reviews do not have an equivalent either, as there is rather little evaluation of the work process itself. Only once during the observation period did Tribeflame have a stand-up meeting that could be interpreted as something akin to the daily Scrum meeting.

Whereas the concrete rules and practices of Agile development do not correspond to Tribeflame's process very well, the more abstract concepts of Agile fit much better. As an example, let us compare the process to the Scrum principles: transparency, inspection and adaption.

Scrum demands that the process is so completely transparent that it is visible at all times who is working on what, and that this is repeated verbally every day at the Scrum meeting. Tribeflame's process is not quite as transparent as that. However, since the developers sit so close to each other and interact so frequently, they are generally well informed about what the others are doing, and they do not hesitate to ask each other what they are working on. Perhaps more importantly, the game they are developing is tested and played constantly, so that everyone knows the current state of the product. This creates a high level of transparency, since there are only a few aspects of the product whose state cannot be assessed by playing the game: namely integration with the distribution channel (Apple's Game Center), and cross-platform support.

Inspection is closely linked to transparency. Inspection in Tribeflame is also performed by playing the game, both by the developers themselves and by external persons. In addition, there are evaluations (Figure 4.2) and frequent discussions (Figure 2.5, page 39) which often include aspects of inspection. That the company instituted daily meetings in the middle of the observation period gives evidence that inspection of the work process itself also happens.

Adaption is very prominent in the work process. There are no de-

tailed, long-term plans; rather, the planning is done as the game progresses from week to week. Features are constantly taken up for discussion at meetings, and whenever a feature in the game is complete it is promptly evaluated in discussions and has an effect on further development. As mentioned above, daily meetings are added halfway through the observation; this shows that adaption is present at the process level, as well as the product level.

Thus, the principles of Scrum correspond well to Tribeflame's process. The common values of Agile development, as expressed in the Agile manifesto, correspond partly to the process. We recall the Agile values, shown in the manifesto reproduced in Figure 3.8 (page 67):

1. *Individuals and interactions* over processes and tools.
2. *Working software* over comprehensive documentation.
3. *Customer collaboration* over contract negotiation.
4. *Responding to change* over following a plan.

Regarding the first value, at Tribeflame, individuals and interactions are clearly much more important than processes and tools, as processes are seldom referred to, and tools are only referred to when necessary. Regarding the second value, Tribeflame's product is always in the form of working software, and the company produces very little documentation in general, and no documentation that can be called comprehensive. Regarding the fourth value, the decision process at Tribeflame is dynamic and adaptive rather than planned in advance.

The third value is not applicable to Tribeflame. The company produces for a mass market of anonymous players, so they do not collaborate with customers or with representatives of customers in the sense that is meant in Agile development. Nor do they enter into contracts or negotiate them with customers: they have investors, but their role is not quite the same as that of a customer. This value therefore seems to reflect an aspect of Agile development that is absent in Tribeflame.

As we can see, many, but not all, of the values of Agile development correspond well to Tribeflame's process. With one exception, the Agile concepts are well suited, in contrast to the concepts used by software engineering which, as we have seen, are in fundamental conflict with Tribeflame's way of working.

However, though Agile concepts are generally compatible with Tribeflame's process when used descriptively, they have serious deficiencies as analytical concepts. The reason for this is that even though the Agile concepts are based on a philosophical ethical system, as described in Section 3.2.5 (Work ethics), the concepts themselves are primarily part of a prescriptive system of software development. Agile methodologies are possible approaches to productive software development, but they are not well suited to analysing processes that are not Agile or intended to become Agile.

Thus, the Agile concepts offer us no help in explaining the differences we see between Agile processes and Tribeflame's process. For example, Agile concepts cannot explain why the playing of other companies' games is so important to Tribeflame. Agile thinking certainly leaves room for this activity, but it offers no guidance in determining what its function is.

Another example is the absence of a customer or customer representative in Tribeflame's process. The customer (represented in Scrum by the Product Owner) is such a central part of Agile thinking that its absence in Tribeflame's process is something of a mystery from an Agile perspective.

4.1.3 *Computer science*

As we saw in Section 3.3 (Computer science), mainstream computer science perceives itself as a mix of applied mathematics and formal

methods. During the observation period at Tribeflame, mathematics was applied in only two cases, both of which had to do with how the graphics should be moved around on the screen in the game, and used mathematics of a high school level. There were no cases where formal methods were applied, or could appropriately have been applied.

A common view of programming within computer science is that a program consists of algorithms and data structures. The task of the programmer is to divide the program into modules, then use the best algorithms he can find. Tribeflame, like any other software company, needs to select algorithms for its programs. However, it is not a top priority that the algorithms are the best possible – they merely have to be good enough.

As an example: before developing one of their games, the developers were worried whether the main algorithm was too inefficient to run on an iPad. They quickly developed a simple prototype of the algorithm and let it run with a few hundred objects, far more than would typically used in a real computer game. When this test proved successful, they proceeded with development without caring further about the efficiency of the algorithm. No mathematical analysis or advanced methods were employed apart from this simple but effective practical test.

Another common computer science view of programs is to regard them as computable functions. This means that there is some input and a mathematical rule that transforms it into output. This view does not correspond very well to the practice of Tribeflame either, for the reason that it is unknown what form the input and output should take for a computer game. If those working at Tribeflame spent their time figuring out how a game could be represented mathematically, they would not have time actually to make the game.

As we have seen, in computer science there is a pronounced orientation towards the machine, and not towards the uses to which it is

put. This perspective is not consistent with the practice of Tribeflame. On one hand, the machine is central to the work, because without it there could be no computer games at all, and the limitations of what the people can do are dictated by the limitations of the machine. On the other hand, though the limitations of the machine are certainly present in everything they do, they are seldom discussed or addressed directly. The discourse at Tribeflame normally focuses not on the machine itself, but on the experience of using the machine. This is not a technical but a human discourse, and it takes place mostly in terms of human imagination and emotions.

If we look at the computer science perception of the work process involved in programming, we see that the correspondence with Tribeflame's process is mixed. There is within computer science an emphasis on mathematical intellectual elegance in the work process. Since mathematics plays no big role in Tribeflame's work, this emphasis is obviously lacking in their process. Computer science also emphasizes the individual in the process, almost to the point of becoming personal. Neither does this emphasis resonate with Tribeflame's process, which is intensively social and cooperative.

On the other hand, computer science emphasizes that the details of a problem only becomes understood through actual programming. This corresponds well to Tribeflame's process, which is essentially a continuous cycle of discussing ideas and then trying them out in practice: that is, programming them. There is also Naur's view, that programming is essentially a process of achieving a certain kind of insight. This corresponds well with the ongoing knowledge building in Tribeflame, which we will examine in more detail in the next section.

4.2 *Hermeneutical analysis*

4.2.1 *Short introduction to hermeneutics*

Hermeneutics is a theory both of how understanding takes place and of what understanding is. In this section, some concepts from hermeneutics will be applied in an analysis of Tribeflame's work process, in order to interpret what the developers do and to understand better why the work process takes the form that it does.

Figure 4.3 lists some key hermeneutical concepts that will be applied in the analysis, along with some very short explanations of the concepts. The use of each concept and its more precise meaning will become more clear in the analysis itself. The interested reader can find a more detailed explanation of hermeneutical theory in Chapter 9 (Understanding).

4.2.2 *The work process*

When we looked at Tribeflame's process we noticed the rhythm inherent in the work. There is a continuous rhythmical shift between working at the individual desks in the periphery of the room, and coming together at the central table to have discussions. At the same time, the rhythmical shift is between working with concentration and alone, on one hand, and on the other hand talking, bantering, joking, and interacting with others. The work that is mostly done alone at individual desks is specific tasks. Through the tasks the game is slowly built. Each task amounts to doing something, trying something out, and getting experience when it is seen if and how it works. In the

Hermeneutical concept	**Characteristics**
Prejudice	Prejudice is a prerequisite for understanding – it can have either a positive or negative influence.
Authority	These are the sources of reason that are recognized as valid.
Received tradition	All understanding builds upon some kind of tradition, and carries this tradition onwards itself while also contributing to it.
Personal reverence	This is the basis for authority. Since understanding is done by persons, personal reverence is crucial.
Pre-understanding	This is the kind of factual understanding that is necessary for understanding, but leaves relatively little room for interpretation – for example to know the language of a text one wishes to read.
Understanding	In hermeneutical theory, understanding is productive and existential. Briefly put, this means that understanding has consequences.
Hermeneutical circle	A concept that expresses the relationship between, for example, part and whole, or between action and reflection.
Effective history	A concept that expresses the historical nature of knowledge and understanding.
Question	All interpretation and understanding is driven by some kind of question.
Horizon of understanding	One's horizon of understanding is the amount of things that are understandable given the present state of one's knowledge and prejudice. Phenomena beyond the horizon of understanding appear meaningless.
Application	Since understanding has consequences, it is crucial to what end the understanding takes place. The intended application of an effort of understanding has an impact on the understanding itself.

Figure 4.3 – Some hermeneutical concepts used in the analysis of Tribeflame's game development process.

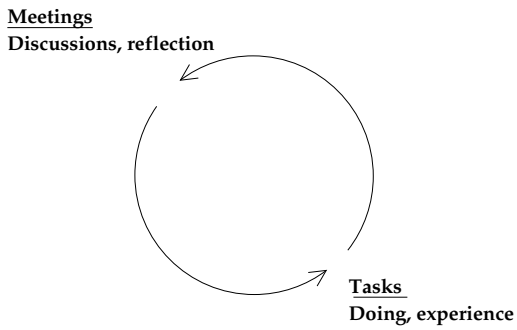

Figure 4.4 – The hermeneutical circle depicts the relationship between tasks and meetings. Tasks are usually carried out alone and result in experience. In meetings the experience from the tasks is discussed and reflected upon. Meetings and tasks are mutually dependent.

meetings the experience is discussed and reflected upon. The meetings provide afterthought, which again leads to ideas for new tasks that can be tried out.

The relationship between meetings and tasks is shown schematically in Figure 4.4 as a circle that leads from meetings to tasks, and back again from tasks to meetings. Meetings and tasks are dependent on each other. The meetings generate the ideas and plans for tasks that should be carried out. The tasks provide experience of what works and what does not, and serve as input to the meetings to make sure that the discussions are grounded in reality. It would be extremely inefficient to discuss and plan the whole game in detail before attempting to carry it out in practice; likewise, it would be unwise to make a whole game without stopping along the way to reflect on the progress.

This sort of mutual dependence is very common in hermeneutics,

and is called the hermeneutical circle.[1] The hermeneutical circle expresses that experience and reflection cannot be separated. Neither can be subordinate to the other, and they need to happen concurrently.

Because the hermeneutical circle is such a common phenomenon, it is often also recognized outside of hermeneutic theory, under another name. Donald Schön has analysed the practice of a range of modern professions: engineers, architects, managers, and so on. He writes that the professional oscillates between involvement and detachment, from a tentative adoption of strategy to eventual commitment:[2] "The unique and uncertain situation comes to be understood through the attempt to change it, and changed through the attempt to understand it."[3] We recognize here the structure of the hermeneutical circle as it is observed in Tribeflame.

In theory, this rhythmical, circular process goes on indefinitely. In practice, the game is deemed finished at some point. The meetings and tasks do not stay the same but lead to an improving understanding of the game being developed. Schematically, we can picture the relationship between tasks and meetings in the hermeneutical circle as time goes by as the spiral shown in Figure 4.5. The hermeneutical circle is still seen shifting from tasks to meetings and back again, but we also see that the process becomes more focused with time.

Initially, the process is very open. The available alternatives are large and visionary, and the chosen decisions are not clung to with great commitment. But as time passes and understanding deepens, decisions made in the past become increasingly costly to revise be-

[1] The hermeneutical circle depicted here shows the relationship between experience and reflection. The hermeneutical circle also appears between other concepts. The most common use of the hermeneutical circle is to describe the relationship between the part and the whole in a process of understanding: see Section 9.4 (The concept of understanding).

[2] Schön 1983 p. 102.

[3] Ibid. p. 132.

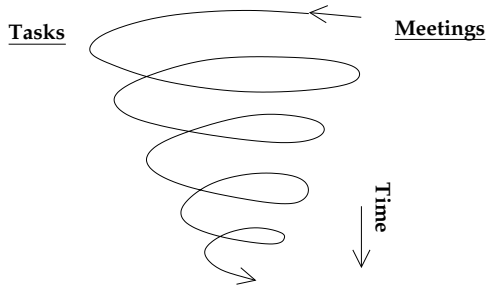

Figure 4.5 – Over time, the hermeneutical circle between tasks and meetings results in better understanding. Whereas the process is very open in the beginning and decisions are relatively easy to undo, with time it becomes more and more tight and resistant to radical change.

cause time and resources have already been committed to them. Decisions are still up for discussion – the process is not rigid – but it is no longer so easy to undo them. The process has become more tight, and focused. Schön recognises the same gradual tightening of the professional process. He writes that the professional can always break open his chosen view, but that "This becomes more difficult to do as the process continues. His choices become more committing; his moves, more nearly irreversible."[1]

4.2.3 *Horizons of understanding*

Tribeflame's process is not simply a matter of planning, optimizing, or construction. It is primarily a process of understanding. At the outset, the developers do not fully understand the myriad details that

[1] Ibid. p. 165.

together make up a successful game. As the work progresses, they slowly and gradually come to a fuller understanding of the central question of their work: what makes a game fun, and specifically, what makes the game we are working on right now fun?

The process of understanding does not start from scratch. Tribeflame's team is composed of individuals with different and highly specialized competences. Matti and Kati are competent in the area of graphic art; Andreas, Mickie, and Fredrik in the area of programming. Björn is competent in the areas of business economy and programming. They all have a general understanding and personal experience of computer games. All this foundational knowledge and competence forms the pre-understanding that is the basis of the process of understanding. If Tribeflame's developers did not have this pre-understanding, they would first have to acquire it before the process could begin of understanding the game they are developing.

The understanding process is a social process that takes place in meetings, discussions, and chat between the developers.[1] As we have seen in Figure 2.8 (page 41), "information sharing" in the sense that some information is simply delivered from one person to another is quite rare – the process of understanding is much more interactive. Nor is this primarily a decision-making process. Decisions are usually the outcome of discussions, but the discussions contain much more than the information and deliberations needed to make decisions. Sometimes, the decisions that are the result of a discussion are left unstated. As soon as the developers reach a common understanding, the consequences often become so clear to them that everyone agrees on the common direction.

So how can we characterize the content of the discussions, if it is neither information sharing nor explicit decision making? The developers seek to expand their understanding in the process. At the on-

[1] See Section 9.6 (Horizons of understanding).

set of a discussion, they have a certain knowledge and range of ideas that they are prepared to understand: their horizon of understanding. Through discussions they exchange facts, opinions, and viewpoints, and in that way expand their horizon. The process of understanding is a process of horizon fusion. When the developers' horizons of understanding fuse, their understanding becomes larger – not by streamlining everyone's opinion, but by expanding each individual's horizon.

4.2.4 *Authority and tradition*

Though the process of understanding is complex and difficult to describe in simple categories such as, for example, the software engineering phases, this does not mean that it is unstructured or haphazard. The process is strongly guided by authority in various forms. The final authority of decisions rests with the owners, Björn and Andreas. They share the responsibility for business decisions, while Björn has the overall product responsibility, and Andreas has the overall technical responsibility. The business decisions rarely impact upon the development process directly, and in this area they exercise their authority freely. Regarding the development, they rarely exercise their authority directly, as we saw in Chapter 2 (Programming practice), with Björn's reluctance to impose a decision outside of the consensus process. It does happen, however. For example, Andreas has insisted that the programmers use shared pointers, a form of memory management.[1]

In their daily operation and management of business decisions,

[1]The class shared_ptr of the Interprocess library from the Boost collection of libraries for C++ (2011).

there is hardly any conflict between the two. The basis of the unproblematic sharing of responsibility and authority between them is their good relationship – the personal reverence they have toward each other. The authority they have over their employees is based partly on the formal economic relationship between employer and employee, and partly on the personal reverence the employees have for Björn and Andreas in Tribeflame's small, tight-knit team.

Since Tribeflame's process is primarily a consensus process, the dominating form of authority in discussions is not that of Björn and Andreas as employers. The work is a teamwork of understanding. Each individual has a specialized role and authority within his field of competence, but the work is not only a collaboration between experts where each has the final say in his or her area: rather, each individual can influence every area of the game and contributes to the overall direction of the development. The primary form of authority between the developers comes from the ability to argue convincingly.

Convincing arguments also rest on authority and, as mentioned above, not merely the authority of expertise. The most frequently invoked authority in Tribeflame's discussions is that of other games. All the developers have a notion of what a good game should be like, as well as examples of games that they consider successful. An argument about how their own game should be is very often supported by a reference to another game that contains the feature in question, and thus the other game lends its success to the argument.

A few games carry so much authority that a reference to them can almost settle an argument. The most authoritative game is *Angry Birds*, the most successful tablet game in the world, which is also made by a Finnish company. *Cut the Rope* is also mentioned often, perhaps because its gameplay is somewhat similar to Tribeflame's own game. Older games are also mentioned occasionally, for example *The Incredible Machine* from the 1990s.

The collected knowledge of the developers about how a game is supposed to be forms a tradition with which they became acquainted before they became involved with Tribeflame, and which continues to develop. Tribeflame's games are built upon, and are a continuation of, this tradition of computer games.

The tradition of computer games is an important source of authority, and this is one reason why the Tribeflame developers spend so much time playing games and talking about them. Another is that the computer game tradition provides the yardstick with which the success of Tribeflame's games is measured. As long as the developers do not have hard data on their game's success in the form of sales figures or number of downloads, they have to compare their half-finished game to the existing tradition. For this reason, the fusion of horizons of understanding that goes on in the hermeneutical circle is not only a fusion of the horizons of the individual developers on the team; it is at the same time a fusion of the developers' horizons and the horizon of tradition.

Thus there are at least two levels of fusion going on in the hermeneutical process of Tribeflame: a fusion of the different expertises that are needed to make a game, and a fusion with the tradition that sets the standard for what a good game is. With this notion, we can explain some aspects of Tribeflame's process that were difficult to analyse satisfactorily with mainstream theories of programming. Tribeflame's developers spend a significant amount of time playing games made by other companies (see Figure 2.10, page 43), which has the effect both of providing them with arguments and of making them more knowledgeable of the computer game tradition.

They also spend a significant amount of time playing their own game (see Figure 2.9, page 42). This has not only the function of finding errors in the programming and graphics, but also gives the developers a sense of whether their game is fun to play: that is, whether

they are reaching their goal. Playing their own game is not only a technical activity that corrects the code: it is as much or more a way for the developers to get a sense of the practical application of their game. They are trying to put themselves in the place of the eventual player, and experience the game as a player would.

To experience the game as another person would requires that the developer can suspend his horizon of understanding temporarily, or else his own knowledge will prevent him from understanding the other person's experience. For example, the developer must try to forget his knowledge of how a certain puzzle is meant to be solved, and approach the puzzle as if he saw the game for the first time, as an eventual player. This is a difficult task, and to make it easier the developers also have other people from outside the company play their game (see Figure 2.9).

The developers experience their game on the basis of prejudice. In hermeneutic theory, prejudice does not have the negative association that it does in everyday language. Rather, prejudice is a requirement for understanding – understanding simply cannot happen without prejudice, whether the prejudice is consciously known or not. Prejudice is simply the fundamental assumptions humans make in order to make sense of things.

However, prejudice can be either true or false. True prejudice will make understanding easier, while false prejudice will hinder understanding. To examine, and correct, prejudice is a central part of the process of understanding. The developers do this by playing their game themselves, but correcting one's own prejudice is difficult precisely because it is the basis of understanding. This is an area where the developers cannot trust their own competence completely.

Therefore it is of great importance to them to have external people play their game. It provides them with an interpretation of the game that they have not made themselves, and that they can use to correct

their prejudice and move along in the process of understanding.

4.2.5 *Having fun*

All the activities at Tribeflame, and the process of understanding that is analysed above, are directed by the application of their product: to play the game and have fun. From the use of shared pointers in the code to the choice of colour scheme for the graphics, and even to business decisions such as an eventual merger with another company – every decision has to be evaluated in the light of the overall goal of creating a game that is fun to play. The developers are conscious of this when they say that the code is just a means to reach a goal.[1] This has an important consequence for programming research, in that it does not make sense to study programming in isolation from the context in which it appears. Computer game code cannot be understood without some knowledge of computer games.

The application, in the form of "having fun", is what drives the activities, but the application in itself is not quite enough to explain the process of understanding fully. After all, if "fun" were completely defined by an existing computer game, all Tribeflame would have to do was to copy that game. All understanding is in essence driven by a question, and for Tribeflame the question is what "fun" actually means. The beginning of Tribeflame's answer is that "fun" means entertainment.[2] This might seem self-evident, but it is not. Consider, for example, that for Björn and Andreas themselves, "fun" means to work, among other things.[3]

[1] See note 2, page 44.

[2] See note 3, page 44.

[3] Andreas: "Is it already time for lunch?" Björn: "Yes, time passes quickly when you're having fun." Field diary, Monday 5th September 2011, 11:47.

Thus, in essence, Tribeflame's business is a process of trying to answer the question of what it means to have fun, for the kind of person that would be inclined to play its game; and all of the company's activities are somehow related to this question. Of course, those working at Tribeflame have to do many activities that do not directly contribute to answering this question, such as administration of the payroll. But the supporting activities are only necessary for success, not sufficient. They are not the critical factor that decides whether Tribeflame will succeed or not in competition with other competent companies.

In the analysis of Tribeflame with hermeneutical theory, we have seen the use of all the essential hermeneutical concepts apart from one: the concept of effective history. Effective history means to be conscious of the role of the knowledge one produces in a historical perspective. During the observation period, Andreas touches upon this question a single time when he muses on the fairness of charging money for computer games.[1] His comment brings to mind the question of the moral justification for the game, and of the place of their product – entertainment – in the grander scheme of things. But his is only an offhand comment; an explicit consciousness of effective history is absent in Tribeflame's process.

4.2.6 *Summary*

The process of Tribeflame can be explained as a hermeneutical process, which is shown in schematic form in Figure 4.6. The alternating activities of doing tasks and discussing them makes up a hermeneutical circle in which understanding is built through repetition. The increase in understanding takes the form of a fusion of horizons of

[1] See note 4, page 44.

Hermeneutical concept	**Results**
Prejudice	The developers' ideas of what the game players will like. Corrected by external playtests.
Authority	Business decision authority rests with the owners. Other authority comes from convincing arguments, drawing from expertise and references to exemplary games.
Received tradition	An informal knowledge of computer games from life experience. Rests on a collection of authoritative games.
Personal reverence	The developers respect each other's competence and personal authority.
Pre-understanding	Skills and expertise in programming and graphic design.
Understanding	A gradual process that slowly deepens over time, and that involves the whole company.
Hermeneutical circle	Manifest in the mutual dependency between tasks that bring experience, and meetings that reflect upon the experience.
Effective history	Largely absent.
Question	The question of what "fun" actually means. All activities relate to this.
Horizon of understanding	Expanded during discussions in the company. Developers' horizons fuse with each other, and with computer game tradition.
Application	To make a game that is fun to play, in order to provide entertainment, so that the company can stay in business.

Figure 4.6 – A schematic summary of the principal results of analysing Tribeflame's process with hermeneutical theory.

understanding, both between the individual developers, and between the developers as a team and the outside world: their customers and competitors.

The hermeneutical process rests on a foundation of pre-understanding and prejudice. The pre-understanding is made up of the developers' pre-existing skills and competences. The prejudice is made up of their ideas of what a computer game is supposed to be like, and what it means for it to be fun.

The process is guided by authority and tradition. Authority draws on various sources, and derives both from the expertise of the developers and from the tradition of computer games. The tradition is not ancient and static; rather it is living and constantly evolving. Thus, the newest games on the market are also part of tradition.

The process is directed by the goal of the computer game – its application. This is the reason for the process, and as such it influences all parts of the process. The reason – entertainment – is an encompassing perspective that must be kept in mind in order to understand the process and the product correctly.

Bibliography

Abrahamsson, Pekka, Outi Salo, Jussi Ronkainen, and Juhani Warsta 2002. *Agile Software Development Methods: Review and Analysis.* VTT Publications 478. VTT Technical Research Centre of Finland. Espoo. ISBN 951-38-6010-8.— *page 65.*

Agile Alliance 2001. *Manifesto for Agile Software Development.* http://agilemanifesto.org. — *pages 65, 104.*

Aiken, Howard H. 1964 [1975]. "Proposed Automatic Calculating Machine." Previously unpublished memorandum. *IEEE Spectrum* 62–69. In: Brian Randell (editor) 1975. *The Origins of Digital Computers: Selected Papers.* Second edition. Springer-Verlag Berlin Heidelberg New York. ISBN 3-540-07114-8.— *page 83.*

Aranda, Jorge 2010. *A Theory of Shared Understanding for Software Organizations.* PhD thesis. Graduate Department of Computer Science, University of Toronto.— *page 22.*

Auvinen, Jussi, Rasmus Back, Jeanette Heidenberg, Piia Hirkman, and Luka Milovanov 2006. "Software Process Improvement with Agile Practices in a Large Telecom Company." In: Jürgen Münch, Matias Vierimaa (editors). *Product-Focused Software Process Improvement.* Proceedings of the 7th International Conference on Product Focused Software Process Improvement. Lecture Notes in Computer Science vol. 4034. Springer-Verlag Berlin Heidelberg. ISBN 978-3-540-34683-8.— *page 65.*

Beck, Kent 1999. "Embracing Change with Extreme Programming." *Computer* vol. 32, issue 10. IEEE computer Society Press. Los Alamitos, California.— *page 78*.

Beck, Kent, and Cynthia Andres 2004. *Extreme Programming Explained: Embrace Change.* Second edition. Addison-Wesley. Boston. ISBN 0-321-27865-8.— *pages 72, 102*.

Boehm, Barry W. 1988. "A Spiral Model of Software Development and Enhancement." *Computer* vol. 21, issue 5. IEEE Computer Society.— *pages 54, 56, 57*.

Boehm, Barry W. 2006. "A View of 20th and 21st Century Software Engineering." In: Proceedings of the 28th International Conference on Software Engineering (ICSE '06). ACM. New York. ISBN 1-59593-375-1.— *page 47*.

Boehm, Barry W., and Richard Turner 2003. "Using Risk to Balance Agile and Plan-Driven Methods." *Computer* vol. 36, issue 6. IEEE Computer Society Press. Los Alamitos, California.— *pages 79, 80*.

Brooks, Frederick P., Jr. 1975 [1995]. *The Mythical Man-Month: Essays on Software Engineering.* Anniversary edition 1995. Addison-Wesley. ISBN 0-201-83595-9. — *pages 16, 48, 49, 52, 81, 218*.

Brooks, Frederick P., Jr. 1986 [1995]. "No Silver Bullet—Essence and Accidents of Software Engineering." *Information Processing, IFIP.* Elsevier. Reprinted 1987 in IEEE *Computer* magazine vol. 20. In: Brooks 1975 [1995].— *page 62*.

Bruegge, Bernd, and Allen H. Dutoit 2000 [2010]. *Object-Oriented Software Engineering: Using UML, Patterns, and Java.* Third edition, International edition 2010. Pearson Education publishing as Prentice Hall. ISBN 978-0-13-815221-5.— *page 17*.

Bryant, Antony 2000. "It's Engineering Jim ... But Not as We Know It: Software Engineering — Solution To the Software Crisis, Or Part of the Problem?" In: Proceedings of the 22nd International conference on Software engineering (ICSE '00). ACM. New York. ISBN 1-58113-206-9.— *pages 161, 190*.

Buxton, J.N., and B. Randell (editors) 1970. *Software Engineering Techniques.* Report on a conference sponsored by the NATO Science Committee Rome, Italy, 27th to 31st October 1969. NATO Scientific Affairs Division. Brussels.— *page 47*.

Cardwell, Donald 1994. *The Fontana History of Technology.* FontanaPress. London. ISBN 0-00-686176-8.— *page 145*.

Christensen, Lone Rahbek 1987. *Hver vore veje: Livsformer, familietyper & kvindeliv.* Museum Tusculanums Forlag. Københavns Universitet. ISBN 87-7289-243-9.— *pages 23, 25*.

von Clausewitz, Carl 1832-1834 [2007]. *Vom Kriege.* Translated by Michael Howard and Peter Paret. *On War.* Princeton University Press 1976. Abridged by Beatrice Heuser. Oxford University Press 2007. ISBN 978-0-19-954002-0. — *page 242*.

Coase, Ronald H. 1937. "The Nature of the Firm." *Economica* vol. 4, no. 16.— *page 186*.

Cockburn, Alistair 2001. *Agile Software Development.* Addison-Wesley Professional. ISBN 0-201-69969-9. Draft version: 3b.— *pages 65, 66, 72, 73, 74, 75, 80, 82*.

Crawford, Chris 2002. *The Art of Interactive Design: A Euphonious and Illuminating Guide to Building Successful Software.* No Starch Press. San Fransisco. ISBN 1-886411-84-0.— *page 189*.

Czarnecki, Krysztof, and Ulrich W. Eisenecker 2000. *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley Professional. ISBN: 978-0-2013-0977-5.— *page 82*.

DeMarco, Tom, and Timothy Lister 1987 [1999]. *Peopleware: Productive Projects and Teams.* Second edition 1999. Dorset House Publishing. ISBN 978-0-932633-43-9.— *pages 17, 64, 72.*

Dijkstra, Edsger W. 1975. *Homo Cogitans: A Small Study of the Art of Thinking.* Unpublished manuscript no. EWD533. The Center for American History, The University of Texas at Austin.— *page 93.*

Dybå, Tore, and Torgeir Dingsøyr 2008. "Empirical Studies of Agile Software Development: A Systematic Review." *Information and Software Technology* vol. 50, issues 9–10. Elsevier.— *page 65.*

Ehn, Billy 1981. *Arbetets flytande gränser: En fabriksstudie.* Bokförlaget Prisma. Stockholm. ISBN 91-518-1434-X.— *page 24.*

Fein, Louis 1959. "The Role of the University in Computers, Data Processing, and Related Fields." *Communications of the ACM* vol. 2, no. 9. — *page 83.*

Filinski, Andrzej, Robert Glück, and Neil Jones (editors) 2005. *Noter i Datalogi V — Programmeringssprog.* HCØ Tryk. Datalogisk Institut, Københavns Universitet. ISBN 87-7834-663-0. In English.— *pages 87, 89, 91, 93.*

Fishman, Charles 1997. "They Write the Right Stuff." *Fast Company* vol. 6, issue Dec 1996 / Jan 1997: article no. 28121. Fast Company, Inc.— *page 151.*

Fowler, Martin, and Jim Highsmith 2001. "The Agile Manifesto." *Dr. Dobb's Journal.* http://www.drdobbs.com/open-source/the-agile-manifesto/184414755. — *pages 65, 66, 72, 104.*

Gadamer, Hans-Georg 1960 [1965]. *Wahrheit und Methode: Grundzüge einer philosophischen Hermeneutik.* J.C.B. Mohr (Paul Siebeck). Tübingen. Second edition 1965.— *pages 221, 223, 224, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243.*

Gadamer, Hans-Georg 1976. *Rhetorik und Hermeneutik: Als öffentlicher Vortrag der Jungius-Gesellschaft der Wissenschaften gehalten am 22. 6. 1976 in Hamburg.* Joachim Jungius-Gesellschaft der Wissenschaften. Vandenhoeck & Ruprecht. Göttingen. ISBN 3-525-85553-2.— *pages 188, 205.*

Gormsen, Gudrun 1982. "Hedebonden: Studier i gårdmand Peder Knudsens dagbog 1829–1857." *Folk og Kultur.* Offprint in the series *IEF småskrifter.* With an English summary.— *page 24.*

Grady, Robert B. 1997. *Successful Software Process Improvement.* Hewlett-Packard Professional Books. Prentice Hall PTR. Upper Saddle River, New Jersey. ISBN 0-13-626623-1.— *page 57.*

Hedegaard Jensen, P. 1969 [1970]. "Numerisk styring." In: William Cauchi and Hjalmar Petersen (editors). *Bogen om EDB: Databehandlingens hvem–hvad–hvor.* Second revised edition 1970. Politikens Forlag. København. ISBN 87-567-1377-0.— *page 4.*

Heidenberg, Jeanette 2011. *Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches.* PhD Thesis. Department of Information Technologies, Åbo Akademi. ISBN 978-952-12-2533-8. — *page 79.*

Highsmith, Jim 2002. "What is Agile Software Development?" *CrossTalk, The Journal of Defense Software Engineering* vol. 15, no. 10.— *pages 72, 75, 79.*

Hoare, C.A.R. 1980 [1981]. "The Emperor's Old clothes, ACM Turing Award Lecture." *Communications of the ACM* vol. 24, no. 2, 1981.— *page 63.*

Hunt, Andrew, and David Thomas 1999. *The Pragmatic Programmer: From Journeyman to Master.* Addison-Wesley. ISBN 0-201-61622-X.— *pages 72, 76.*

Huth, Michael, and Mark Ryan 2000 [2004]. *Logic in Computer Science: Modelling and Reasoning About Systems.* Second edition 2004. Cambridge University Press. ISBN 0-521-54310-X.— *pages 85, 91.*

Højrup, Ole 1967. *Landbokvinden: Rok og kærne. Grovbrød og vadmel.* Nationalmuseet. København.— *page 24*.

Højrup, Thomas 1995. *Omkring livsformsanalysens udvikling.* Museum Tusculanums Forlag. Københavns Universitet. ISBN 87-7289-337-0. With an English summary.— *pages 156, 157*.

Højrup, Thomas 2002. *Dannelsens dialektik: Etnologiske udfordringer til det glemte folk*. Museum Tusculanums Forlag. Københavns Universitet. ISBN 87-7289-785-6.— *page 157*.

Institute of Electrical and Electronics Engineers 1985. *Computer* vol. 18, no. 8. Theme: "Visual Programming." IEEE Computer Society.— *page 93*.

Institute of Electrical and Electronics Engineers 1990. *IEEE Standard Glossary of Software Engineering Terms.* IEEE Std 610-12.1990. ISBN 1-55937-067-X.— *page 19*.

International Electrotechnical Commission 2009. *Functional safety of electrical / electronic / programmable electronic safety-related systems.* IEC 61508-(1–7) Ed. 2.0. 65A/548/FDIS Final Draft International Standard, distributed on 2009-12-18.— *pages 70, 124, 136, 140, 142*.

Jacobson, Ivar, Shihong Huang, Mira Kajko-Matsson, Paul McMahon, and Ed Seymour 2012. "Semat — Three Year Vision." *Programming and Computer Software* vol. 38, no. 1. Pleiades Publishing, Ltd.— *page 161*.

Jones, Neil D., and David A. Schmidt 1980. "Compiler Generation from Denotational Semantics." In: Neil D. Jones (editor). *Semantics-Directed Compiler Generation, Proceedings of a Workshop*. Springer-Verlag London. ISBN 3-540-10250-7. — *page 93*.

Kennedy, George A. 1984. *New Testament Interpretation through Rhetorical Criticism*. The University of North Carolina Press. Chapel Hill and London. ISBN 978-0-8078-4120-4.— *pages 197, 199, 205, 207*.

Kernighan, Brian W., and Dennis M. Ritchie 1978 [1988]. *The C Programming Language.* Second edition 1988. Prentice Hall PTR. Englewood Cliffs, New Jersey. ISBN 0-13-110362-8.— *page 86.*

Knuth, Donald E. 1968-2011. *The Art of Computer Programming.* Vol. 1, 1968 [1997]. *Fundamental Algorithms.* Vol. 2, 1969 [1997]. *Seminumerical Algorithms.* Vol. 3, 1973 [1998]. *Sorting and Searching.* Vol. 4A, 2011. *Combinatorial Algorithms.* Addison-Wesley. Reading, Massachusetts, and Upper Saddle River, New Jersey. ISBN 0-201-89683-4; 0-201-89684-2; 0-201-89685-0; 0-201-03804-8.— *page 95.*

Knuth, Donald E. 1984. "Literate Programming." *The Computer Journal* vol. 27, no. 2. British Computer Society. Oxford University Press. — *page 189.*

Knuth, Donald E. 1989 [1990]. "The Errors of TEX." *Journal of Software: Practice & Experience* vol. 19, no. 7. John Wiley & Sons, Ltd. In: Tom DeMarco and Timothy Lister (editors) 1990. *Software State-Of-The-Art: Selected Papers.* Dorset House Publishing. New York. ISBN 0-932633-14-5.— *page 95.*

Knuth, Donald E. 1992. *Literate Programming.* Center for the Study of Language and Information. Leland Stanford Junior University. ISBN 0-937073-80-6.— *page 95.*

Lammers, Susan 1986. *Programmers at Work.* Microsoft Press. Redmond, Washington. ISBN 0-914845-71-3.— *pages 86, 94.*

Lange, Martin 2012. "Integration nicht-sicherer Software in sicherheitsgerichtete Systeme." 10th International TÜV Rheinland Symposium, May 15–16. Cologne.— *page 149.*

Latour, Bruno 1987. *Science in Action: How to Follow Scientists and Engineers Through Society.* Harvard University Press. Cambridge, Massachusetts. ISBN 0-674-79291-2.— *pages 22, 27, 91, 214.*

Latour, Bruno, and Steve Woolgar 1979 [1986]. *Laboratory Life: The Construction of Scientific Facts.* Sage Publications, Inc. Second edition 1986. Princeton University Press. Princeton, New Jersey. ISBN 0-691-02832-X.— *page 207.*

Lauesen, Søren (Soren) 2002. *Software Requirements: Styles and Techniques.* Addison-Wesley. ISBN 978-0-201-74570-2.— *pages 17, 62, 63, 163.*

Liddell, Henry George, and Robert Scott 1940. *A Greek-English Lexicon.* Revised and augmented throughout by Sir Henry Stuart Jones with the assistance of Roderick McKenzie. Clarendon Press. Oxford.— *page 27.*

Mahoney, Michael S. 1990. "The Roots of Software Engineering." *CWI Quarterly* vol. 3, no. 4.— *page 83.*

Mallery, John C., Roger Hurwitz, and Gavan Duffy 1987. "Hermeneutics: From Textual Explication to Computer Understanding?" Massachusetts Institute of Technology Artificial Intelligence Laboratory. In: Stuart C. Shapiro (editor). *The Encyclopedia of Artificial Intelligence.* John Wiley & Sons. New York.— *page 21.*

McConnell, Steven C. 1993. *Code Complete: A Practical Handbook of Software Construction.* Microsoft Press. Redmond, Washington. ISBN 1-55615-484-4.— *pages 189, 206.*

Medoff, Michael D., and Rainer I. Faller 2010. *Functional Safety: An IEC 61508 SIL 3 Compliant Development Process.* Exida.com L.L.C. Sellersville, Pennsylvania. ISBN 978-0-9727234-8-0.— *pages 142, 145.*

Naur, Peter 1985 [2001]. "Programming as Theory Building." Reprinted 1992 in *Computing: A Human Activity.* In: Cockburn 2001.— *pages 81, 95.*

Naur, Peter 1995. *Datalogi som videnskab.* DIKU Rapport no. 95/4. DIKU Tryk. Datalogisk Institut, Københavns Universitet. ISSN 0107-8283.— *pages 87, 92.*

Naur, Peter, and Brian Randell (editors) 1969. *Software Engineering.* Report on a conference sponsored by the NATO Science Committee. Garmisch, Germany, 7th to 11th October 1968. NATO Scientific Affairs Division. Brussels.— *page 47.*

Patterson, David A., and John L. Hennessy 1997. *Computer Organization and Design: The Hardware / Software Interface.* Second edition. Morgan Kaufmann Publishers, Inc. San Fransisco. ISBN 1-55860-491-X.— *pages 48, 87, 88.*

Paulson, Lawrence C. 1991 [1996]. *ML for the Working Programmer.* Second edition 1996. Cambridge University Press. ISBN 0-521-56543-X.— *pages 84, 91.*

Peirce, Charles Sanders 1905 [1931–1958]. *Collected Papers of Charles Sanders Peirce.* Vols. 1–6 edited by Charles Hartshorne and Paul Weiss. Vols. 7–8 edited by Arthur W. Burks. 1931–1958. Harvard University Press. Cambridge, Massachusetts. — *page 228.*

Perelman, Chaïm, and Lucie Olbrechts-Tyteca 1958 [2008]. *Traité de l'argumentation: La nouvelle rhétorique.* Sixth edition 2008. Editions de l'Université de Bruxelles. ISBN 978-2-8004-1398-3.— *pages 190, 191, 199, 204.*

Perlow, Leslie A. 1997. *Finding Time: How Corporations, Individuals, and Families Can Benefit from New Work Practices.* Cornell University Press. Ithaca and London. ISBN 978-0-8014-8445-2.— *page 217.*

Perlow, Leslie A. 1999. "The Time Famine: Toward a Sociology of Work Time." *Administrative Science Quarterly* vol. 44, no. 1. Sage Journals.— *page 217.*

Pierce, Benjamin C. 2002. *Types and Programming Languages.* The MIT Press. Cambridge, Massachusetts, and London. ISBN 0-262-16209-1.— *pages 85, 89.*

Reynolds, Carl H. 1970. "What's Wrong with Computer Programming Management?" In: Weinwurm. — *page 16*.

Rorty, Amelie Oksenberg 1983. "Experiments in Philosophical Genre: Descartes' Meditations." *Critical Inquiry* vol. 9, no. 3.— *page 223*.

Royce, Winston W. 1970. "Managing the Development of Large Software Systems." *Proc. IEEE WESCON*. IEEE Press.— *pages 55, 63*.

Rönkkö, Mikko, and Juhana Peltonen 2012. *Software Industry Survey 2012*. School of Science, Aalto University. Espoo. ISBN 978-952-60-3614-4.— *pages 33, 101*.

Scanlon, Leo J. 1981. *The 68000: Principles and Programming*. Howard W. Sams & Co., Inc. Indianapolis, Indiana. ISBN 0-672-21853-4.— *page 84*.

Schön, Donald A. 1983. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books. United States of America. ISBN 0-465-06878-2.— *page 111*.

Schwaber, Ken 2003. *Agile Project Management with Scrum*. Microsoft Press. Redmond, Washington. ISBN 0-7356-1993-X.— *pages 69, 70, 73, 74, 75, 77, 79*.

Schwaber, Ken, and Jeff Sutherland 2010. *Scrum Guide*. http://www.scrum.org.— *page 73*.

Sirelius, Uuno Taavi 1906–1908 [2009]. *Suomalaisten kalastus I–III*. Suomalaisen Kirjallisuuden Seura. Helsinki. Reprinted 2009. Suomalaisen Kirjallisuuden Seuran Toimituksia 1252. ISBN 978-952-222-152-0.— *page 24*.

Solin, Ulla 1992. *Animation of Parallel Algorithms*. PhD thesis. Department of Computer Science, Åbo Akademi. Acta Academia Aboensis, Ser. B, Mathematica et physica, vol. 52, no. 2. Åbo Akademis förlag. ISBN 952-9616-03-1.— *page 92*.

Spolsky, Joel 2004. *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity.* Apress. ISBN 978-1-59-059-389-9.— *pages 24, 72.*

Stroustrup, Bjarne 1985 [1997]. *The C++ Programming Language.* Third edition 1997. Addison-Wesley. ISBN 0-201-88954-4. — *pages 86, 89, 94, 203.*

Suenson, Espen 2013. "Method and Fieldwork in a Hermeneutical Perspective." In: Frog and Pauliina Latvala (editors). *Approaching Methodology.* Humaniora 368. Finnish Academy of Science and Letters. ISBN 978-951-41-1085-6.— *pages 32, 125.*

Turkle, Sherry 1984 [2005]. *The Second Self: Computers and the Human Spirit.* Simon & Schuster, Inc. New York. Twentieth Anniversary edition 2005. MIT Press. Cambridge, Massachusetts, and London. ISBN 978-0-262-70111-2.— *page 25.*

U.S. Defense Science Board 1987. *Report of the Defense Science Board Task Force on Military Software.* Office of the Under Secretary of Defense for Acquisition. Washington, D.C.— *page 47.*

U.S. Department of Commerce 2011. GDP by Industry. Tables of Value Added by Industry. Bureau of Economic Analysis. — *page 18.*

U.S. Department of Defense 1985. *Military Standard: Defense System Software Development.* DOD-STD-2167. Washington, D.C.— *page 70.*

Vincenti, Walter Guido 1990 [1993]. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History.* Johns Hopkins Paperbacks edition 1993. The Johns Hopkins University Press. Baltimore and London. ISBN 0-8018-4588-2.— *pages 20, 53.*

van Vliet, Hans 2008. *Software Engineering: Principles and Practice.* Third edition. John Wiley & Sons, Ltd. ISBN 978-0-470-03146-9.— *page 18.*

Weber, Max 1922 [2003]. "Kapitel IX. Soziologie der Herrschaft. 2. Abschnitt. Wesen, Voraussetzungen und Entfaltung der büreaukratischen Herrschaft." First published in: *Wirtschaft und Gesellschaft. Dritter Teil. Typen der Herrschaft.* J.C.B. Mohr (Paul Siebeck). Tübingen. Translated from: Marianne Weber and Johannes Winckelmann (editors) 1990. *Wirtschaft und Gesellschaft. Grundriss der verstehenden Soziologie. Zweiter Teil.* J.C.B. Mohr (Paul Siebeck). Tübingen. In: Heine Andersen, Hans Henrik Bruun, and Lars Bo Kaspersen (editors) 2003. *Max Weber: Udvalgte tekster. Bind 2.* Translated by Ole Bjerg. Hans Reitzels Forlag. København. ISBN 87-412-2554-6.— *pages 81, 144.*

Weinwurm, George F. (editor) 1970. *On the Management of Computer Programming.* Auerbach Publishers Inc. United States of America. Standard Book Number 87769-044-8.— *page 47.*

Willim, Robert 2002. *Framtid.nu: Flyt och friktion i ett snabbt företag.* Brutus Östlings Bokförlag Symposion. Stockholm and Stehag. ISBN 91-7139-549-0. With an English summary.— *page 35.*

Wilson, Kax 1979 [1982]. *A History of Textiles.* Paberback edition 1982. Westview Press. Boulder, Colorado. ISBN 0-86531-368-7. — *page 27.*

Winskel, Glynn 1993. *The Formal Semantics of Programming Languages: An Introduction.* Foundation of Computing Series. The MIT Press. Cambridge, Massachusetts, and London. ISBN 0-262-73103-7. — *page 87.*