Organizing Simulation Code Collectives

Mikaela Sundberg

This article examines the ways researchers develop and use computer programs for numerical simulations and the different social relationships that are involved in creating the frames for these activities. On the basis of ethnographic case studies of numerical simulation practice in astrophysics, oceanography, and meteorology, including climate modelling, the present article discusses how work with simulation codes can be discussed by means of a typology of simulation code collectives. This typology provides a systematic account of how a particular and increasingly important form of software development and use takes place in science. It also contributes to current discussions on the relation between producers and users of technology by suggesting that the definition of them as empirical categories can be understood through the social relationships that the people (simulationists) working with them are embedded in.

Keywords: simulation codes, organization, user/developer distinction

Numerical simulations have become important and widely used to perform scientific work, not least in natural scientific fields where traditional experiments are unattainable. This is the case in meteorology, oceanography, and astrophysics. Temporal and spatial scales of phenomena such as galaxy formation, ocean currents, or climate change make them impossible to bring into a laboratory, but as mathematical representations they can be brought into computers.[1] It is when they appear in the form of computer programs that these models can generate numerical simulations and new scientific knowledge.[2] Numerical models can therefore be understood as codes (cf. Mackenzie, 2006). How scientists work with and relate to them is an important aspect in reaching a better understanding of contemporary, computer-based scientific practice.

Simulation codes are tricky to develop and work with. It often takes a long time to develop a code that does not crash during calculations and when sufficiently stabile, a code is still sensitive to how the "numerical experiments" conducted with it is set up in terms of initial conditions, parameter settings etc. (see e.g. Winsberg, 2003; Kennefick, 2001). In addition, output is often difficult to evaluate and this is one of the reasons why simulationists with similar scientific interests not only compete to produce the most interesting and innovative results, but also collaborate to agree upon standard results for certain scientific problems (Sundberg 2008; forthcoming). Many simulationists use codes they have

built themselves, and their professional reputations are tied to the simulation codes they have developed (cf. Lahsen, 2005). When codes are accessible for others, for example, by being possible to download from a webpage, developers are no longer in control of how "their" codes are applied. Considering this lack of control in relation to the difficulties in using simulation codes in general, it becomes interesting to investigate the different ways the simulation codes are distributed and shared and the possibilities of organizing and controlling this venture.

Literature on software distinguishes among first, open software where coding is visible and proprietary software where coding is kept secret and second, among free and commercial software (see e.g. Fuller, 2003). Free open source software has enrolled huge amounts of software developers and the phenomenon has been much debated among the practitioners themselves (Mackenzie, 2006: 69). Sociological studies are less common, but they have, among other things, investigated the dynamics within groups of people developing and using a particular code or operational system (e.g. Ratto, 2007; Stewart, 2005; von Krogh et al. 2003). [3] Sociological interest has propagated from workers and production to consumers and consumption and likewise, STS has shifted focus from production and producers of technology to use and users of technology. This has led to problematizations of previous conceptions of users and designers and the boundary between them. [4] Information technology is an area in which this question becomes highlighted, in part due to the proliferation of free open source software.

Model developers are most of the time also some kind of users (Lahsen, 2005), but the distinction between developers (authors/producers/designers) and users have nevertheless been used to differentiate between types of simulationists (see e. g. Dowling, 1999; Merz, 1999; Sundberg, 2005).[5] In addition, "developer" and "user" are important empirical categories that simulationists use to construct their social world (cf. Mulkay and Gilbert 1982). There is a salient discourse defining simulationists as either "developers" or "users". "Developers" focus on the inside of the code and try to understand it, whereas "users" approach codes as black-boxes (cf. Woolgar, 1991). From a user perspective, code development is time-consuming and costly, especially since code development is often only reported in technical reports. To publish results based on simulations is a more prestigious activity. Why invent the wheel twice when someone else provides the technology needed to "do science"? Free codes are readily applicable after the set up of initial (and occasionally boundary) conditions and usually a few alterations of particular parameter values. More extensive adjustments or additional components are necessary when available codes do not provide exactly what is required for a particular application. Also simulationists who take their point of departure in a code constructed by someone else may therefore develop a particular piece for their own use, while relying for the existing framework for the rest. This implies–not surprisingly–that the boundary between the activities of development and usage, and between researchers who develop or use, is unclear in practice.

The present article aims to explain how development and use of simulation codes, and the roles related to these activities, can be understood by analyzing

the different ways that work with codes is organized. By means of a typology, the article suggests how simulationists together with the codes they work with constitute different *simulation code collectives*. With a focus on organization rather than the content of scientific knowledge, the article analyzes how simulation code collectives are organized and what implications this has for the definition and control of simulation code use and development. The purpose is also to lift the gaze from the level of single projects or programs and provide basis for some more general conclusions. This is achieved by presenting a typology that informs further investigation of numerical simulations in science, and possibly also software development in general.

Next, an introduction to the typology clarifies its theoretical basis in organizational theory. This is followed by a short section on methodology, including a presentation of the three ethnographic case studies of scientific numerical simulation practice in astrophysics, meteorology, and oceanography that constitute the basis for the analysis. Then I present each category of the typology and position them in relation to technology studies on users and developers as well as to literature on open source software. In the concluding part, I summarize the analysis and discuss some themes that illustrate the usefulness of the typology in generating questions that would promote our understanding of how numerical simulations take place in science.

Organizing User and Developer Relations - An Ideal Typology of Code Collectives

In her analysis of high energy physics experiments as super organisms, Knorr Cetina (1995) draws attention to the detector as a central and centring object within the communal life-form. Without implying that simulation codes hold the same central place–which they rarely do–Knorr Cetina's discussion nevertheless provides a basis for talking of users and developers of a particular simulation code as constituting a form of collective. This is also similar to how Collins (1985: chap 3) used the so-called TEA set to define a set of scientists with one problem in common, but others not in common. The notion of simulation code collectives is an attempt to capture the organizational aspects of technology (code) use, rather than the question of what knowledge that is required to develop and use it (see e.g. Winsberg, 1999; 2003). The analysis will nevertheless touch upon the problems with replicating simulation codes–to make them work also when used by others than the original developer (cf. Collins 1985, chap 3).

A simulation code collective consists of simulationists who work with a particular code. These simulationists do not have to consider themselves to be a group of some kind. The defining characteristic of this form of collective is that all its simulationists work with, and are therefore related through, the same code. The relationships between the simulationists and between the simulationists and the code may be of different kind. I focus on how it differs in terms of organization.

The construction of an ideal typology of code collectives helps us to discuss how work with simulation codes is differently organized and what consequences this has for how use and development is defined. Ideal types are theoretical constructions that do not have to have correspondance in any concrete

empirical instance. They serve as means to help in abstraction, without loosing the meaning of the actors (Weber, [1904] 1949).

Use/ Development	Open	Closed
Free	Code developed all around	Code spread all around
Limited	Code of the group	Code of the centre

Table 1 A typology of simulation code collectives.

In this typology, exhibited in table 1.1, the use dimension has two categories. These categories refer to whether a code is accessible for every researcher who would like to use it for numerical simulations (free) or whether access is limited and the code is then only available for a certain group of people (restricted). The development dimension is constructed in the same way. Is any user welcome to make changes inside the code and add features to it or are there restrictions concerning this activity? Thus, code collectives are not formal organizations, yet I will explain how they can be understood in relation to organizational elements.

Ahrne and Brunsson (2009) emphasize the importance of partial organization and how attempts to create order through organizing can be found outside and between formal organizations through use of some, but not all, elements of organization. Membership, hierarchy, rules, monitoring, and sanctions are such organizational elements. *Membership* entails obtaining a certain identity (as member), and therefore as expecting to be treated by the organization in a way that differs from non-members.[6] *Hierarchy* is the right to decide over others, where the source of power is a decision. Members have to comply with *rules*, which are mostly written and always pronounced. An organization has the right to *monitor* compliance with them and also to bring *sanctions*. Furthermore, Ahrne and Brunsson (2009) contrast organization, and the use of organizational elements, with the original sense of the term network as referring to personal and informal relations. A network which introduces organizational elements acquires an altered status.[7]

How can we understand use and development of simulation codes in relation to organizational elements (or the absence of them)? In organizational terms, free or restricted refer to whether membership is required, or expected, for certain activities (use or development). Membership is the most important organizational element in the typology, but in practice, it is combined with others. These elements, as well as certain limitations of the typology, are further discussed in the sections on particular types of collectives. The order of presentation is based on increasing inclusion of organizational elements into the collectives.

Methodological Considerations

The typology is based on three case studies of numerical simulation practice within meteorology, oceanography, and astrophysics. The primary material consists of interviews and observations, mainly carried out in Sweden. I have interviewed scientists who work with numerical simulations in the fields of astrophysics (eleven interviews), meteorology (seven interviews), and oceanography (twelve interviews).[8] Nowadays, both atmospheric and ocean modelling is often part of

coupled climate modelling exercises, where ocean and atmospheric models are connected. Five of the ocean simulationists and six of the meteorological simulationists worked with an ocean/atmospheric component of a climate model. Participant observation has been conducted during about 25 seminars, one workshop, three conferences, and two code user meetings. [9] I also draw upon additional material from an earlier study of meteorological research (see Sundberg, 2005). This includes ten interviews with meteorological simulationists at a research department and five interviews at the Swedish weather service. I recorded the former, but only took notes during the latter.

Secondary material consists of information presented at the official web pages (including user's guides, manuals, reports) of certain codes and e-mail lists. This material covers 13 astrophysics codes, six meteorological codes, seven ocean codes, and two coupled climate models. Most of these are codes that my informants work with, but a few additional codes have also been selected. The reason for these additional codes is that they appear to be among the most widely known and used codes in the different communities (when these were not already included). Most of these widely used codes are obviously developed outside of Sweden, particularly in the US. Whereas interviews and observations primarily informs about perspectives, common knowledge, and practice–the actors' level of meaning–, public material informs about official guidelines, recommendations, and rules–important organizational elements.

The typology evolved from a hybrid approach that combined a data-driven approach with a theory-driven approach (Boyatzis 1998: 51ff.). This involved following an inductive approach in identifying themes, but also the use of theories to guide the articulation of meaningful themes. Thus, the typology grew out of a three-stage process involving a inductive analysis of primary material, where I classified all material on collaborative work with simulation codes and distinguished among free use and free development and developed analytical codes (themes) regarding the conditions and organization of these collaborations. I then used basic organizational concepts to further clarify dimensions of a typology and used secondary material for further development. Since it is an ideal typology, a few code collectives fit into a combination of two types rather than one particular. I use accounts on code work as well as descriptions from web pages to illustrate the different types. Importantly, my aim is not to create types that characterize code work in each discipline, but to present types that describe general as well as specific patterns. Thus, quotes from researchers are primarily used to illustrate features of collectives, rather than as representative accounts from a researcher with a particular disciplinary affiliation.

A final, methodological question concerns how to distinguish between different codes. This will be done based on the names that the codes have acquired (cf. Bezroukov, 1999). Different names do not reflect the theoretical content and underlying premises of the codes. Many codes in a given research field are based on the same equations, but they differ in terms of, for example, the numerical method/technique they employ, programming conventions, and formulation of sub-grid parameterizations (cf. Edwards, 2001;

Sundberg, 2009; Merz, 1999). Codes exist in several versions or generations, often distinguished by numbering (1.0, 1.1, 2.0 etc). Nevertheless, names are more than simple reference markers. They point at the social identities of codes (cf. Ahrne and Brunsson, 2008: 93ff.) and simulationists gather around codes as an effect of their name.

The Code of the Group

Although I am not describing the dynamics of collectives in terms of how they develop from fitting into one type to fitting into another, it is worth pointing out that the code of the group collective can be seen as the most original type of collective. It also appears as the most common type. It is therefore natural to start the presentation here.

The code of the group collective is closed for researchers outside of the research group and there is no user which has not also developed and improved the code. Development is free within the collective, not among simulationists in general. When you work within the code of the group, you have to be (or have been) a developer to be a user. The latter role presupposes the former role. Everyone in the collective has this double relation to the code.

A single simulationist is responsible for the birth of the code and the collective grows if s/he collaborates with colleagues–researchers and/or doctoral students–to further develop the code. The number of people working with each code is small and the codes which fit into this category are rarely well-known outside the group, perhaps in part because the network-character of the collective makes it quite invisible from the outside (cf. Ahrne and Brunsson, 2009). Thus, this type of code does not belong among those in each field that researchers outside the collective gossip about (these seem to be quite few). Put differently, the codes themselves do not have any reputations, but the researchers who work with them may have.

Because work conducted by this type of code collective is unrelated to formal organizational boundaries, the simulationists in the collective can bring their code with them to new organizational affiliations and continue development and use wherever they want (as long as they do not move into an organization where work is dedicated to a particular code). This happens when simulationists move from one department to another, especially in the early stage of their careers when they finish their PhD's or post doc period. The consequence is that if there is no explicit attempt to streamline development, codes end up being developed in different ways. Everyone works on a particular version of the code and all of them are equal. There is no status difference between the various versions.

Even if there are no formal restrictions that deter researchers from sharing their codes with others, there are other reasons not to do it. One astrophysicist, who has developed several codes and who identifies himself as "clearly in the kind of developing code, developing algorithm side", said the following:

> The people who write codes don't... very few take the *trouble of making it such so that you can easily give it to other people to use*. Partly this takes a lot of time. Also because it's actually quite hard to make something which works on *very many different problems* without…There is a little bit the feeling that you have to really know what you are doing before you use the codes. There is a psychological barrier almost, you don't want to… *You don't want to*

> *take the responsibility that people do stupid things with your code*. And so there are only a few groups that really have come out with codes that many people use. (Emphasis added)

This account implies that there is a fear of making code available to others. One reason for being afraid of that users "do stupid things with your code" (whatever that means) is that it might affect the reputation of the developer of the code. The developer is identified with the code by others and also identifies him/herself with it, including on an emotional level (cf. Lahsen, 2005). This suggests the importance of controlling users in different ways. This is further discussed in relation to the free codes. The account also suggests how additional work is required to make the code user-friendly ("easily give it to other people"), for example, by writing understandable comments in the code and some form of user's manual. There is less written documentation to be found for the code in the code of the group type of collective because user-friendliness is not an aim. To make computer programs esoteric–for others–makes them "private" (cf. Ratto, 2007: 79). To keep codes esoteric could therefore also be a strategy to *maintain* them "private" and prevent other simulationists from using it, even if they have access to it.

Moreover, the code in the code of the group collective tends to be tailored for the particular purposes of the group, where all work on similar scientific problems (cf. Mattila, 2006). To make a code applicable to a wider variety of problems requires expertise in several research areas, beyond the focus and competence of a small research group. To give an example, one meteorological simulationist spoke of a code he had developed together with others:

> This model that we have developed ourselves…you discover after a while that the model has its limitations and if you want to move beyond those limitations, then it has to become a more advanced model system. And if you are going to have a more advanced model system, you have to keep it *alive* and update it with new things as soon as science progress, because otherwise, you are *shut out* due to the development. And finally you realize that a small university group can't take care of updating all aspects of such a model. And then it is pretty attractive to collaborate with a large centre, where there is a model developed… And then you get regular updates to the model. (cf. Mackenzie 2006: chap 4)

This account illustrates how a small university group may have difficulties to develop their simulation code with the novel insights that simulationists in their field are expected to include (otherwise you are "shut out").[10] This problem can be solved by becoming a user of a free code, where others provide "regular updates". The quoted meteorologist decided to switch to a public code and concluded regarding the code he abandoned: "It will probably die as time goes by, when those who originally developed it pass away in one or the other way, or change model, like we did." Since the code in the code of the group collective never leaves the hands of its developers, it "dies" when its developers/users abandons it and perhaps move on to work with a different code. Lindsay (2003: 50) remarks that the disappearance of a technology from the public does not necessarily end the life of that technology, but this requires a "public/private" distinction that does not exist in the code of the group collective.

The Code developed all around

This is the type of collective which has most similarities with what is elsewhere referred to as free open source software. Perhaps surprisingly, very few of the codes in my material fit into the ideal typical construction. The reasons for this will be discussed below. Because of their rarity, I use work with the one suitable code in astrophysics as a running example to characterize the code developed all around collective.

One of the explicit reasons for creating this astrophysics code, which I fictively refer to as the Neptune code, was the aim to make everyone who worked with the code deal with one single version and constantly implement their new developments there for the benefit of all. The aim to stick to one common, yet constantly developed, version is part of the reason why this code is maintained at a central repository, located at a particular server, rather than organized as a distributed system that easily leads to different versions at different places. Versioning systems themselves both reflect and affect how computer program development is organized, but it is beyond the analytical focus of this paper to explore this further.

User meetings, to which everyone in the collective is welcome, are of great importance for the code developed all around. It is during these meetings that the simulationists who work with the code actually meet and discuss face-to-face. In all the other collectives, at least a fraction of the simulationists meet, especially if the organization they work for is situated at a particular location. Because the code developed all around has no formal organizational ties to place, it is not evident where to hold the user meetings. The location shifts. However, the term *user* meeting does not adequately reflect what these meetings are about in the sense that they are as much, or even more, about discussing developments and maintenance, rather than discussing problems among users. This also differs from the code spread all around collective, where user meetings are likely to be more user-oriented, also including messages from developers to users rather than discussions on development. Why this is the case will be explained in the section on the code spread all around collective.

According to one of the founders, about 400 different researchers have "checked out" the Neptune code, but there are only about 25 researchers who have the possibility to "check in" or "commit" changes to the central repository. One of the founders of the code occasionally hands out new passwords in order to grant users the benefit of becoming developers. This gate-keeping function maintains the boundary between developers and users and evokes Raymond's (1999) "cathedral"-model (with a hierarchical structure and a master architect) of open source software. The otherwise relative unstructured and open nature of the Neptune code development fits with the "bazaar"-model. Notwithstanding, distinctions between participants are made all the time, for example, the presentation of the last annual meeting (2008) distinguished between "core-developers", "regular users" and "new users".[11] Different types of engagement are also manifested in how participants take part in discussions. Whereas some engage in discussions on maintenance and code structure, others are only attentive (or present) when discussions of scientific applications are on the agenda.

However, who is a developer and who is a user is not fixed within the collective,

but changes over time (cf. Lindsay, 2003: 43ff.). Whether someone is recognized as or identify him/herself as belonging to one or the other category has nothing to do with formal organizational affiliation (membership). This differentiates the code developed all around from the code spread all around. Any user may become a developer within the existing collective by proposing and providing, for example, additional equations to be implemented. It is then up to the "core developers" in general and the gate-keeper in particular to accept or reject the suggestion. There is therefore some coordination of development by new contributors that regular developers are free from and this, in combination with the gate-keeping function, illustrates a hierarchy.

Let me provide an example of how users become developers, and how this is unrelated to formal affiliation (as well as rank). During one of the annual user meetings, a master's student proposed some developments of the code. After his presentation, the gate-keeper commented on one of them by saying that "I think this equation makes perfect sense and should be implemented". (Yet he also showed his superior expertise by noting that another suggestion was actually "already in the code"). Perhaps this (relative) openness is one of the reasons behind a particular mechanism–organizational element–used to control the development of the Neptune code: In order to monitor changes, the code runs several auto-tests overnight and developers are encouraged to do auto-tests before they check-in changes. The most important aspect in relation to the construction of the type is that in this sense, control focus on *what* is developed rather than *who* does it.

The Code spread all around

> There are those who sit and develop models for the sake of model development, to make it as good as possible at describing the world, and of course they sit and poke. Then there are those of us who sit and use those models, which those others have made, to try to find something out. And it is hard to be both of these two persons. Because it requires so much to know how such a large and encompassing model works. So either you develop models or you use them. And it depends on where you are. I can't sit here and think that I will develop the [research institute] model. Then I would have to be there, with the group who's doing it.

This account is from an ocean simulationist who uses a free code and works outside the research institute where it is developed. It illustrates the common view that simulationists are either developers *or* users. The reason given for this division is the amount of knowledge it takes to know a code, but also the difference in interests between on the one hand, trying to make the code as good a representation as possible and tinker a lot with the code in order to do so (as the developers at the mentioned research institute do), and on the other hand, use an application of a code to tackle some research problem (as the cited ocean simulationist does). Importantly, this ocean simulationist connects the different roles of developer and user to *where* one works. This is a feature of the code spread all around collective.

The code in the code spread all around type of collective is available for non-members to use. There are different degrees of freedom to use however, ranging

from lack of restrictions and possibility to do "what you want" to more restricted utilization with attempts to control use. There are sometimes requirements on user registration, it is impossible to download older versions, and/or refusals to support users of older versions. However, the code is primarily *developed* within a particular organization. This is presented at web pages by phrases such as that the code is "developed at" or that a particular department is the "home" of the code. This type of code is also often talked about as the code of the name of the organization where it is developed, even if the code itself has another name. For example, MOM (the Modular Ocean Model) is often referred to as "the GFDL-model", where GFDL stands for Geophysical Fluid Dynamics Laboratory (Princeton, US).

What makes simulationists enter an open collective? Simulationists mention several factors that they believe to be important for making a free code popular in terms of its amount of users. One of the reasons for popularity that they note is the capacity of codes to deal with a multitude of problems, as opposed to being tailored to particular purposes. Modules can be "switched" on or off depending on what type of problem one is interested in dealing with. This creates a higher number of potential users compared to the code of the group. What seems more important however is that codes are "user friendly". "User-friendly" refers to qualities of the *code* such as being well-programmed and well-documented in articles, reports and/or user's manuals. Yet it also refers to qualities of the *collective* because "user-friendly" implies that there is available support such lists of FAQs provided by the supplier of the code, e-mail lists where everybody can ask questions and developers or other users reply, and user meetings. Several user-friendly measures require resources that are probably more available within a formal organization than among a small group of individual simulationists (code of the group collective) or a dispersed group (code developed all around collective).

Interestingly, simulationists have not mentioned the quality of output that codes generate as a reason for using a free code. Several interviews and informal discussions among simulationists indicate that the most popular codes, at least in astrophysics, actually have the *worst* reputation. The following quote from someone who develops his own astrophysics codes refers to users of a free code developed at a particular institute (a code spread all around collective): "[The code] has been very, very popular and people use it without having an idea how it works, but it was set up so that it would be easy to use by people who don't really know how to make their own codes… and it [the code] is actually not so good." This type of talk about users and popular codes is most characteristic among developers/users of other codes. This is not surprising considering that it is particularly in the interest of developers within code of the group collectives to undermine the quality of free codes. Use of free codes degrades the work and efforts of simulationists in the code of the group collectives to make a code for themselves, as well as their skills in making the code produce reasonable results (cf. Collins, 1985: 73f.).

Although simulationists emphasize the importance of choosing the right code for the problem under consideration, most simulationists tend to stick with the same code no matter what scientific problem they address, in part because learning how to handle and understand a

simulation code takes so much time. Yet the presentation of the code of the group collective discussed how simulationists do not always enter a collective for good, but sometimes leave. This is also the case for code spread all around collectives. For example, one ocean simulationist started to use a free code but found errors in it and decided to develop a new code:

> One artifact was that for instance you have fresh water with no salinity and then you have Atlantic water with a salinity of 35,5 PSU, which is the unit for salinity, and when these two water masses meet …you will get a mixture of the water with an intermediate salinity, not higher and you can not have negative salinity, that is impossible. … This is a type of error that can occur in some models. So we saw that occurring and that was the main reason for me to, I could not use it. … We had to invent something to avoid this. So that was the start of my model.

Negative salinity in an ocean model is a clear sign of something being wrong. In the following, I discuss the responsibility of users and developers in relation to achieving acceptable results and fixing errors.[12] Codes available over the internet are supposed to be well-tested, but they are without guarantees. There is always a risk that simulations crash or generate what is considered to be unreasonable output. One way to offer users of free code some kind of comfort is to provide test problems with known solutions, for example, at the webpage where the code is found. These tests are used in order to test the set-up of the code, but since the code has already passed them before, one can also see them as tests of the *users*. "Once a technology is well established and a culture exists about how to use a machine, any failures are more likely to be attributed to the user rather than to the machine". (Pinch, 1993: 37) However, simulation codes are not *that* well-established. It is therefore contested where failures should be attributed. Depending on what is to blame for failures, it is the responsible developer *or* the user who is expected to solve the problem. One astrophysicist talked about the use of free codes in relation to trust in results: "If you haven't worked in detail with the code yourself, you have to be sure that it is good craftwork (laughter), that what you see is not happening because the code is strange, but because nature is strange (laughter)." If there is something one considers erroneous, the astrophysicist said that "you rely on that someone else fixes it, or that you convince someone that it is actually wrong, that it isn't something that I have done, but that it is actually something they have done wrong in the code." In this case, "someone" is someone who has developed the code and who has the skill to deal with the errors. If developers are not convinced that the code rather than the user which is wrong, this type of incident can be seen as representing what is meant with users may "do stupid things with your code". In this case, the "stupid things" might be to generate strange results because of lack of skill to handle the code properly.

This discussion highlights two important differences between the code of the group collective and the code spread all around collective regarding the division between users and developers. First, there is the question of knowledge transfer. What happens when a code moves to a new setting? A crucial difference between the code of the group collective and the code spread all around collective is that in the first case, the unit

of knowledge is intact (cf. Collins, 1985). A simulationist who has developed the code moves with it. In the latter collective, it is only a code which is diffused, without the knowledge (the simulationists) that made it work. Second, there is the question of the division between developer and user. Within the code of the group collective, there is no boundary between users and developers, within the code spread all around, there are attempts to draw a sharp boundary.

Defining Developers in the Code Spread all Around Collective

In the code spread all around collectives, what defines "development" is a question of which changes end up in the *official* version of the code. This is a question of membership. The official version is the version that users outside of the formal organization can download and it is one out of three types of versions of codes in the code spread all around collective. An *unreleased* version is a version that is unavailable for users outside the organization. New releases are rare, as opposed to what seems to be the case with software more generally (Raymond, 1999). This is because codes are thoroughly tested, but also because developers take the opportunity to approach new scientific problems before releasing the new, hopefully better code, to others. Finally, a *modified* version is a version that users outside the organization have adapted (developed) for their particular purposes and it is not spread to others.

Different types of versions have different status. Work with unreleased versions, which eventually become official, defines what development is, at least from the organizational developers' (members') viewpoint. The organization controls the official version. One astrophysicist, who worked with the development of a free code at the centre which is referred to as its "home", talked about how users often adapt the code to their own problems, but that these changes are not on the "same level" and can't be referred to as "developments". The point here is not *what* the "users" actually do, but the fact that internal developers (members of the formal organization) attempt to control what counts as "development" (as opposed to a "modification" for own purposes). Terminology is also somewhat related to which part of the code you develop. The following quote from an ocean simulationist provides an example:

> Some people do…*some* development on things that are very targeted to their project.
>
> But to change the more fundamental or experimenting with more fundamental aspects of the code… there are very few people who would do that. Because things will very easily break… [i]f you make changes to the core numeric, it is a quite finely tuned, everything depends on everything and if you change one thing, the model will most likely be unstable. ... So there will be very few people that will ever touch that part of the code. But then it can be things how to introduce say … radioactive tracers, you need to include in the source terms maybe an age tracer if you look at the radioactive mechanism and some age mechanism to trace it and then you fill this in.[13]

Dividing between core ("fundamental") and periphery ("some source terms") illustrates how some parts of the code are considered more essential than others, therefore distinguishing between different types of development. It seems

like users outside the formal organization are encouraged to use the code off-the-shelf or, at most, make some adjustment. As non-members, they are not expected to develop (cf. Ahrne and Brunsson 2009; Woolgar 1991). For example, one ocean simulationist who used a ready-made code from a well-known research institute said:"They [researchers at an institute] are kind and develop, then they don't want you to poke into it". This account suggests that users should be grateful for what is provided and avoid making changes, partly in order to avoid that something will "break". A working code is in the interest of developers as well as the users.

Another aspect of development is that development within the formal organization is coordinated. At least ideally, there are not several people working independently on improving the same thing. The code is "heading somewhere", as one ocean simulationist expressed himself about the strategies for developing the particular code he worked with. Development work outside the formal organization is generally not mobilized in this endeavour, but may turn out constructive if internal developers (members) take advantage of products from external, expert users (cf. Lindsay, 2003: 38). This possibility is sometimes recognized in, for example, user's guides where rules state how bits and pieces of code should be designed in order to be accepted, listed for example as "[r]equirements that contributed code must meet".[14] There are also examples of research institutes that invite researchers to sign temporary contracts for working on developments in a simulation code developed there (see also Jankovic, 2004: 60). The development of the Community Atmosphere Model at the National Center for Atmospheric Research (Boulder, US) is one such example where temporary members get the opportunity to develop something that may end up inside the official version of the code, rather than simply in a modified version for personal usage. There are also groups which are asked to do developments. One meteorologist told me about the development work that his group did outside the boundaries of the organization responsible for the climate model that they used:

> We started working on aerosol parameterizations for climate models ten years ago and back then [the centre] didn't really have big ambitions in that area, they felt that it was too early. So we were sort of far ahead of them …. but then gradually things have changed because gradually [the centre] has become more ambitious in terms of aerosols climate interactions and have decided that we have to start including a lot of detail of that stuff in our model…. there are several groups that have been involved in sort of an informal working group … and we have been involved in that, so we have participated in some meetings but they have decided that they want to go ahead with implementing *another* approach from another group … So you know, they know what we are doing, we are trying to show them what we are doing and of course we try to make them to use our work, but you know it is up to them.

This quote illustrates how different groups develop similar new modules for the same free code. This leads to competition, especially in meteorology where it is prestigious to have contributed to the development of well-known codes, preferably to operational weather

forecast models. As one meteorologist wrote in her thesis, "possibly the ultimate hope of every atmospheric modeller is to make a contribution to an operational NWP model" (Zagar, 2004: 38). Although the close relationship between basic, meteorological research and weather forecasting (applied meteorology) does not have a correspondance in astrophysics, it is likely that all code developers wish that their development efforts are recognized. To create a new code collective is another way to enable developing users in codes spread all around collectives to become *recognized* as developers. It is common in all three disciplines that well-known and widely spread codes contribute with bits and pieces to other codes, but they also have off-springs (see also Edwards, 2001). Slightly modified codes are given new names and new codes as well as collectives are born (cf. Sundberg, 2009: 173). Re-naming codes is an important part of this identity transformation.[15]

The Code of the Centre

The code of the centre collective has a formal organizational boundary, both in terms of development and use. One meteorologist emphasized that "a model always needs a place"; a place where people know it and work with it continuously. He expressed his skepticism by speaking of it as "risky" to download a code from somewhere, modify it a bit for own purposes, and then use it. He maintained that new modified versions of the code that non-members create are not tested properly, because "users" utilize a code like a "black-box", whereas "developers care about the quality of the model" (cf. Woolgar, 1991). This is not only an example of stereotypical view on users and developers, but also of how the code of the centre collective keeps its code for its members.

The code of the centre collective is a part of a formal organization. In order to work with the code–to enter the collective–one has to work at the particular department or research institute where part of the work is explicitly devoted to the code.[16] If one quits working there, one has to quit working with the code. Some codes that are officially free have such restricted usage that they almost fit better into the category of code of the centre collective than the code spread all around. For example, the licence agreement for "software" developed at the Max Planck Institute for Meteorology in Hamburg, Germany provides an example.[17] Although the codes are free, there are many elements to create order imposed on the users outside the formal organization. There are rules and hierarchies manifested in that error fixes and modifications must be communicated to the coordinator of model development. There is monitoring in the sense that the title and authors of any publication with results from the "software" shall be sent to the coordinator of model development no later than the publication is submitted to a scientific journal. There are sanctions because the rights under the licence agreement terminate automatically without notice if users fail to comply with any terms of the licence. At the same time, software is provided without warranty of any kind.

Being a part of a code of the centre collective is to be a member of an organization, and your affiliation is with the organization, not with a code. Research staff in these organizations may therefore be forced to change code due to decisions made higher up in the organizational hierarchy. I have been told about several such occasions. Yet it is not the code which is the basis for the

organization, but the other way round. Important research centres develop new codes, or, what is more common and import, adjust *old* codes and give them new names. One meteorologist said: "Each institute or group wants to call it something. And then for example climate models, there is the Hadley model, ECHAM, some Japanese models… but if you look carefully at those models they contain elements which are identical."[18] It seems like prestigious research centres want to have their own code for numerical simulations, whether they are made from scratch or not. This is very much the case with climate models, which often consist of a combination (coupling) of existing models of the atmosphere and the ocean, and sometimes also land and ice models.[19] This combination becomes a new system. The last quote also implies the *national* character of research that is highly evident within climate modelling. Climate models are sometimes referred to as the model(s) of a particular country, e.g. the German model, if not their organizational affiliation, e.g. Max Planck institute model. This indicates how climate modelling has become a question of national prestige (cf. Nolin, 1999). For example, at a seminar, the novel effort to develop a European Earth system model was presented as a project with "mostly small European countries that do not have their own climate model". This is another example of the national character of climate modelling.

Although the present article focuses on the distribution and sharing of codes, it is interesting to note how some code of the centre collectives provide accessible output *data*. This is especially the case with the climate modelling work that underlies the reports by IPCCs Working Group I. Analysis of aspects of these huge data sets, requiring several months of supercomputer cluster calculations, have become a common modelling-based way to do research, far outside the code of the centre collectives that produced the data. This means that the analyst is not required to ever touch the simulation codes that generated output data. More generally, from the point of view of political interest and available funding, climate modelling takes place in a very different context compared to astrophysics (but perhaps not compared to space research). The code of the centre dominates climate modelling, but does not exist in astrophysics. In fact, several previous studies of climate modelling have been based on studies of centres, and thereby implicitly of code of the centre collectives (e.g. Shackley, 1999, see also Edwards, 2000). Thus, the code of the centre collective is not a superfluous category with respect to simulation codes in general, but the type is less documented in the present material.

Concluding Remarks

This article has presented an ideal typology of simulation code collectives in science and showed the usefulness of an organizational approach in discussing software technology development, distribution, and use, without forgetting the benefits of a science and technology studies informed perspective. The typology offers a systematic description of different types of code collaborations that exist in astrophysics, meteorology, and oceanography–as opposed to single case studies–and it emphasizes similarities between the research fields rather than their differences. To highlight the organizational aspects, we can summarize how different types of collectives relate to different organizational elements.

Collective/ Organizational element	Code developed all around	Code spread all around	Code of the centre
Membership		Developer	User, developer
Hierarchy	Negotiated distinction between main developers and users	Distinction between developers and users	Management decisions
Rules	Instructions	Instructions, contracts	Decisions
Monitoring	(Automatic)	Direct (of developers)	Management
Sanctions (+/-)	+ Password to make changes - Withdrawn password	+ User –made development incorporated into official version - No user support	+ Promotion, more responsibility - Degradation

Table 2 Code collectives and organizational elements

The code of the group is excluded from the table since it does not exhibit any organizational element. In this sense it as a genuine network, in the original meaning of the concept (Ahrne and Brunsson, 2009). In the code developed all around collective, organizational elements have been added to retain some control of the official version, while still having the possibility to offer any user the benefit of becoming a developer. This is why I suggest that a password permitting changes is a reward (positive sanction) rather than a manifestation of membership. Of course, one can also discuss grades of membership, but that conceptual discussion is beyond the scope of the present article.

The code of the group and the code spread all around collectives fit well with what is going on in all three disciplines and it is those types, as well as their differences, that I have focused mostly on. The code developed all around and the code of the centre fit less well. The code in the code developed all around collective is closest to free open source software and even if the latter is becoming more common in software development more generally (see e.g. Fuller, 2003; Campbell-Kelly, 2003), it is rare for simulation codes in this study. Why is this so? One suggestion for the rarity is the lack of control over the development this form of collective essentially exhibits, in combination with what is at stake in science. As technology, simulation codes have to work (perform calculations until the end), but these results are also supposed to deliver results that simulationists believe in (cf. Sundberg, 2008). Because of the uncertainties involved in using simulation codes, more users do not necessarily improve the reputation of the code, and therefore not the reputation of the developers of the code either. To offer output data, as code of the centre collectives do, is a way of avoiding the trouble of having what some developers view as unskilled users of simulation codes (while probably generating other types of problem instead). The ability to

form a code of the centre type collective is restricted by the amount of resources that are invested in different research fields and their simulation code development (cf. Edwards, 2001: 64). For the sake of comparison, it is a limitation of the typology that it neglects the different research contexts and funding climates that e.g. cosmology simulations and climate model simulations take place within. More generally, how resources for code development (permanent positions, support from engineers and programmers etc.) and use (access to supercomputers etc.) are acquired are important sociological questions for further research.

There are nevertheless several central topics for which the code collective typology itself provides a good starting point. For example, what does the dynamics of code collectives look like in terms of their development trajectories? Interesting topics are also the distribution and development trends of different types of code collectives in a given discipline or research field. This is also a part of understanding software development in general. Would shifts in the distribution of different types of code collectives imply shifts in the role simulation codes play in scientific practice? It is quite remarkable how many simulationists still work with their own codes, considering all free codes, the time it takes to write functioning codes, and the little credit that is given for this development. Because of the risks with supplying free code it is also interesting to investigate why there is work put into making free codes. The more epistemological issues are deliberatively absent from the paper, but this is only the consequence of an attempt to provide a distinctively sociological analysis of simulation code collaboration and work in general, which seems to be backlogging the growing number of philosophical discussions on this form of scientific practice. However, the questions raised above are but some of the important ones for exploring both how numerical simulations are embedded and affect the different epistemic cultures of astrophysics, oceanography, and meteorology as well as how the role of simulation codes develop in science more generally.

Acknowledgments

The author wishes to thank Göran Ahrne, Boel Berner, Fredrik Movitz, Alma Persson, Ebba Sjögren, Jan Tullberg, Sven Widmalm, and two anonymous reviewers for valuable recommendations and comments on earlier versions of this article. The author also acknowledges the generous financial support from the Swedish Science Council under contract 2006-1296 and contract 2007-1627.

Notes

[1] Simulation codes are based on mathematical models that use some numerical time-stepping procedure to obtain the model's behavior over time. Within the physics-based sciences, these simulation codes are often built on the basis of constraint-based equational models derived from physical law. Algorithms formally specify how the models behave and are simulated, but model execution differs from model design. There may be different computer codes based on the same mathematical model. In journal articles based on numerical simulations, the mathematical model is generally described, but less information is provided regarding the

code. The scientists who work with simulation codes are often looser in their terminology when they talk about their work with numerical simulations. Astrophysicists generally speak of "codes", but sometimes of "models" when they refer to how they set up their experiment. Meteorologists and oceanographers talk about "models", even if they obviously refer to a version of a computer program. I do not analyze the different terminologies further and generally refer to "code", as it reflects how numerical simulations rely on computers.

[2] Some suggest that building simulation codes and their underlying models is also a process of knowledge creation (see e.g. Winsberg, 1999), but it is still their output that draws most attention as knowledge producing.

[3] There is also much discussion on the values and political implications of the so-called open source movement (e.g. Feller et al., 2005; McInerney, 2009). Within that context, the distinction between free software and open source software are important, see GNU (http://www.gnu.org/) and Open Source Initiative (http://www.opensource.org/).

[4] See Oudshoorn and Pinch (2007) for a thorough review of founding approaches in STS as well as recent developments in user-technology relationships.

[5] Simulation codes connect people in different ways. Through transformations of theoretical models into mathematical models into computer code their multiple forms bring diverse practices together. Theoreticians provide the conceptual basis and develop equations, applied mathematicians work on the numerical approximations and methods, programmers make the codes more efficient, and experimentalists provide data and knowledge about empirical relationships required to set parameters in the code. In relation to these groups, simulationists are generally users who implement and adjust the work of theoreticians, set-up and run codes, and analyse their output. It is this group of practitioners–the simulationists–who concretely work with simulation codes to produce scientific knowledge and it is therefore this group that the present article focuses on. It does not discuss the contribution of more theoretically or empirically oriented research workers who never touch the numerical model in the form of a computer program nor how distinctions between groups are made.

[6] It does not imply identification with an organization.

[7] The network concept in actor-network theory obviously diverges significantly from the traditional sociological understanding of the concept (see e.g. Latour, 2005). This is not discussed in the present article.

[8] These include one meteorologist in Norway, one meteorologist in the Netherlands, four ocean simulationists in Norway, and three astrophysicists in Denmark and they have been selected as informants through snowball sampling. The remaining interviewees worked in Sweden, but several of the research scientists did their doctoral work elsewhere (e.g. Germany, Finland, the Netherlands, France). 12 of the interviews were conducted in English, the rest in Swedish. Quotes from the latter interviews have been translated.

[9] Most of these gatherings were held at or (co-)organized by the Department of Meteorology at Stockholm University (which also hosts physical oceanography), the Department of Astronomy at Stockholm University, Nordic Institute for Theoretical Physics, and the Department of Physics and Astronomy at Uppsala University. Researchers from abroad were often present.

[10] One reason for these difficulties may be lack of resources and complaints about lack of resources for code development is are commonly heard from all simulationists who work with development.

[11] There were only four and eight participants during the first two meetings, according to the original developer. I attended the third and fourth meeting. The number of participants had increased to about twenty. Several of them characterized themselves as new users.

[12] For analysis of how to determine whether to conceive of output as reasonable or not, see Sundberg (2008).

[13] Ocean (and atmospheric) simulationists have a particular and somewhat confusing way of referring to the "dynamics" and the "physics" of codes. The "core" of dynamical equations is referred to as "dynamics" and other process descriptions (e.g. turbulence, cloud convection) as "physics".

[14] See http://www.gfdl.noaa.gov/~fms/pubrel/j/mom4/doc/mom4_manual.html.

[15] Original developers are still credited. We should distinguish between new conceptual developments, for example, the analytical formulation of a process description and new and/or different ways of coding this formulation. The present article concerns code changes rather than model changes. See also footnote 1.

[16] Weather forecasting takes place within national meteorological offices, which are state agencies. See Fine (2007) for an analysis of the work at weather service offices in the US. In Europe, several weather bureaus cooperate to develop and use particular weather forecast models. These multi-institutional collaborations are better characterized as meta-organizations, with organizations rather than individuals as members (cf. Ahrne and Brunsson, 2008).

[17] See http://www.mpimet.mpg.de/en/wissenschaft/modelle/model-distribution/procedure.html.

[18] The Hadley model refers to a climate model developed by the Meteorological Office Hadley Centre, UK and ECHAM to a climate model developed by Max-Planck-Institute for Meteorology and the Meteorological Institute of Hamburg University.

[19] See Edwards (2000) for the history of atmospheric general circulation modelling and an overview of how different numerical models owe inspiration and pieces of code from others. See also Jankovic (2004).

References

Ahrne, G. & N. Brunsson. (2008) Meta-organizations (Cheltenham, UK, Northampton, MA: Edward Elgar).

Ahrne, G. & N. Brunsson. (2009) 'Organization outside organizations. The significance of partial organization'. Submitted.

Bezroukov, N. (1999) 'Open Source Software Development as a Special Type of Academic Research (Critique

of Vulgar Raymondism)', First Monday 4(10) URL: http://firstmonday.org/issues/issue4_10/bezroukov/inidex.html

Boyatzis, R. E. (1998) Transforming Qualitative Information: Thematic Analysis and Code Development (Thousands Oaks/London / New Delhi: SAGE Publications).

Campbell-Kelly, M. (2003) From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry (Cambridge, MA: MIT Press).

Collins, H. M. (1985) Changing Order. Replication and Induction in Scientific Practice (London: SAGE Publications).

Dowling, D. (1999) 'Experimenting on Theories', Science in Context 12 (2): 261–273.

Fine, G. A. (2007) Authors of the Storm. Meteorologists and the Culture of Prediction (Chicago and London: University of Chicago Press).

Edwards, P.N. (2000) 'A Brief History of Atmospheric General Circulation Modelling', in Randall, D. A. (ed) General Circulation Development, Past Present and Future: The Proceedings of a Symposium in Honour of Akio Arakawa (New York: Academic Press).

Edwards, P.N. (2001) 'Representing the Global Atmosphere: Computer Models, Data, and Knowledge about Climate Change' in C.A. Miller and P.N. Edwards (eds) Changing the Atmosphere: Expert Knowledge and Environmental Governance. (Cambridge, Massachusetts: The MIT Press): 31-66.

Feller, J., B. Fitzgerald, S.A. Hissam & K.R. Lakhani. (2005) Perspectives on Free Open Source Software (Cambridge, MA: MIT Press).

Fuller, M. (2003) Behind the Blip: Essays on the Culture of Software (New York: Autonomedia).

GNU http://www.gnu.org/ 02/13/2009

Jankovic, V. (2004) 'Science Migrations: Mesoscale Weather Prediction from Belgrade to Washington, 1970-2000', Social Studies of Science, 34 (1): 45-75.

Kennefick, D. (2000) 'Star Crushing: Theoretical Practice and the Theoreticians' Regress', Social Studies of Science, 30 (1): 5-40.

Knorr Cetina, K. (1995) 'How Superorganisms Change: Consensus Formation and the Social Ontology of High-Energy Physics Experiments', Social Studies of Science, 25 (1): 119-147

Lahsen, M. (2005) 'Seductive Simulations? Uncertainty Distribution around Climate Models', Social Studies of Science 35(6): 895-922.

Latour, B. (2005) Reassembling the Social. An introduction to Actor-Network-Theory (Oxford: Clarendon).

Lindsay, C. (2003) 'From within the Shadows: Users as Designers, Producers, Marketers, Distributors, and Technical Support', in N. Oudshoorn and T. Pinch (eds.) How Users Matter: The Co-Construction of Users and Technologies (Cambridge, MA: MIT Press): 29-50.

Mackenzie, A. (2006) Cutting Code. Software and Sociality (Peter Lang: New York).

Mattila, E. (2006) Questions to Artificial Nature: A Philosophical Study of Interdisciplinary Models and their Functions in Scientific Practice**.** Doctoral Dissertation, University of Helsinki.

McInerney, P.B. (2009) 'Technology Movements and the Politics of Free Open Source Software', Science, Technology, and Human Values 32 (2): 206-233.

Merz, M. (1999) 'Multiplex and Unfolding: Computer Simulation in Particle Physics', Science in Context, 12 (2): 293–316.

Mulkay, M. and Gilbert, N. (1982) 'Accounting for Error: How Scientists Construct their Social World when they Account for Correct and Incorrect Belief', Sociology 16: 165-183.
Nolin, J. (1999) 'Global Policy and National Research: The International Shaping of Climate Research in four European Union Countries', Minerva 37: 125–40.
Open Source Initiative http://www.opensource.org/ 02/13/ 2009
Oudshoorn, N. & Pinch, T. (2007) 'User-Technology Relationships: Some Recent Developments', in Hackett, E. J, Amsterdamska, O., Lynch, M., Wajcman, J. (eds.) The Handbook of Science and Technology Studies (MIT Press. 3rd edition): 541-566.
Pinch, T. (1993) 'Testing, One, Two, Three-Testing: Towards a Sociology of Testing', Science, Technology, and Human Values 18 (1): 25-41.
Ratto, M. (2007) 'A Practice-Based Model of Access for Science: Linux Kernel Development and Shared Digital Resources', Science Studies, 20 (1): 73-105.
Raymond, E. (1999) The Cathedral and the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary (Beijing: O'Reilly).
Stewart, D. (2005) 'Social Status in an Open-Source Community', American Sociological Review 70 (5): 823-842.
Sundberg, M. (2005) Making Meteorology. Social Relations and Scientific Practice. Stockholm: Stockholm University Studies in Sociology, New Series 25.
Sundberg, M. (2008) 'Socialization into Numerical Simulations: The Perspectives of Simulationists in Astrophysics and Oceanography' Department of Sociology Working Paper Series 2008:14.
Sundberg, M. (2009) 'The Everyday World of Simulation Modelling: The Development of Parameterizations in Meteorology', Science, Technology, and Human Values 32 (2): 162-181.
Von Krogh, G., S. Spaeth, and K.R. Lakhani (2003) 'Community, Joining, and Specialization in Open Source Software Innovation: A case study', Research Policy 32: 1217-1241.
Weber, M. ([1904] 1949) '"Objectivity" in Social Science and Social Policy', The Methodology of the Social Sciences (New York. The Free Press): 49-112.
Winsberg, E. (1999) 'Sanctioning Models: The Epistemology of Models', Science in Context 12 (2): 275–292.
Winsberg, E. (2003) 'Simulated Experiments: Methodology for a Virtual World', Philosophy of Science 70: 105-125.
Woolgar, S. (1991) 'Configuring the User: The Case of Usability Trials', in J. Law (ed) A Sociology of Monsters: Essays on Power, Technology and Domination (London: Routledge): 58-100.
Zagar, N. (2004) Dynamical Aspects of Atmospheric Data Assimilation in the Tropics. Doctoral dissertation, Department of Meteorology, Stockholm University, Stockholm.

Cited sources

http://www.mpimet.mpg.de/en/wissenschaft/modelle/model-distribution/procedure.html (07/10/2008)
http://www.gfdl.noaa.gov/~fms/pubrel/j/mom4/doc/mom4_manual.html (09/18/2007)

Mikaela Sundberg
Department of Sociology
Stockholm University
Universitetsvägen 10B
S-106 91 Stockholm
Sweden
mikaela.sundberg@sociology.su.se