

# Are Systems Engineers Complete Losers When It Comes to Communication?

**Author:** Chris Rupp  
**Keywords:** Communication, NLP  
**Total number of pages:** 8  
**Total number of words:** 3.515  
**Published:** Cutters IT Journal

**SOPHIST GmbH**  
**General Manager: Christine Rupp,**  
Dipl. Information Technology (FH)  
**Roland Ehrlinger**  
**Vordere Cramergasse 13**  
**90478 Nuremberg**  
**Germany**

fon: +49 (0)9 11 40 900-0  
fax: +49 (0)9 11 40 900-99

e-mail: [heureka@sophist.de](mailto:heureka@sophist.de)  
Internet: [www.sophist.de](http://www.sophist.de)

**Copyright © 2009 by SOPHIST GmbH**

This publication is protected by copyright. All rights, also transferring to translation, reprint, as well as the reproduction of certain parts of the publication are reserved. It is prohibited to reproduce, reprint or spread any part of this publication neither by using electronic systems nor by employing any other method. This is valid for teaching purposes as well. It is necessary to obtain a permission in writing! The rights of third parties are not to be affected.

# Are Systems Engineers Complete Losers When It Comes to Communication?

The systems development industry is frequently shaken by negative reports, such as the mishaps and delays that plagued the Toll Collect system in Germany<sup>1</sup> or cars that break down due to software defects. Many people wonder what it is that ails modern systems development. Why is it so hard to build systems that realize user requirements when our technology allows us to do almost anything? Are systems engineers losers when it comes to communicating about the requirements? Can the problem be solved by an agile process and direct communication between system users and developers? If so, which fields of knowledge must be added to an agile process in order to ensure that knowledge transfer works?

In order to understand and solve the problems of systems development, we must face two facts that have a strong impact on the development of technical systems:

1. **The complexity trap.** Software, hardware, and electronics are merging into one system. This means that the complexity of our systems is escalating, almost exponentially.
2. **The communication trap.** Different people from different parts of the business, and people from different disciplines, need to work together on these complex systems. The requirements for the system are usually communicated and documented in natural language — and are often not understood by their intended audience.

## THE COMPLEXITY TRAP

Your car is an excellent example of a complex system, being made up of software, hardware, and electronics. Since 1968 the amount of software in motor vehicles has been rising continuously, and this trend will persist into the future. Figure 1 shows the increase in the automotive software component. The cause of this development is functionality that ensures your safety and riding comfort (antilock braking system, electronic stability program, etc.), as well as navigation and Web information systems. This trend is not confined to the automotive industry; it can also be seen in consumer electronics and control and feedback control systems.

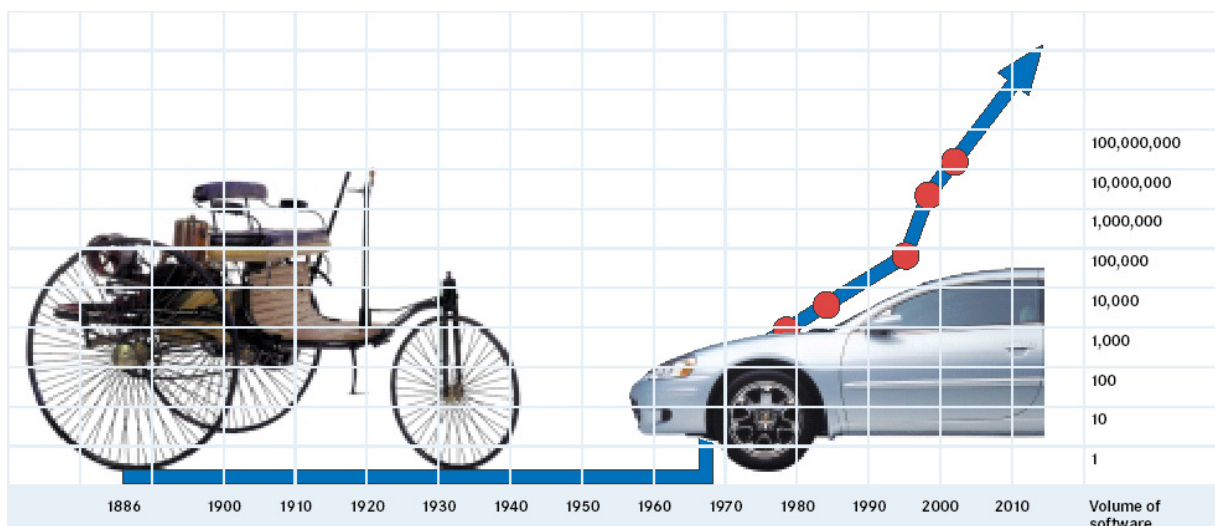


Figure 1 — Growth of software in automobiles. (Source: DaimlerChrysler)

The process of specifying innovative technical systems is difficult. Such systems are complex due to a significant number of embedded systems, which are typically cross-linked. These embedded systems are usually highly complex and handle a large number of exceptions [1]. For example, modern infotainment systems in cars combine telephony, Internet access, radio, MP3, CD, navigation, and road condition and traffic broadcasts, with the possibility of further entertainment media. This is where

specialists of various disciplines, such as mechanical engineering, information technology, business economics, design, and/or electrical engineering come together. Each of them — including the user, about whom we shall not forget — lives and works in his own world. Each of these disciplines brings with it different history, experiences, education, notations, techniques, and, of course, vocabulary. Often systems development is distributed among multiple businesses, which makes contractual interfaces necessary. This “multi-culti,” caused by a system that requires many disciplines for its realization, not only brings about synergies and new opportunities, but also a lot of communication problems.

Up to now, heavy specifications and contracts have been the method of choice for dealing with complex situations. It is no secret that this approach usually results in heavy expense, as it usually involves hundreds of pages of specification and, subsequently, thousands of requirements. The contracts that must be written and negotiated are no less complex. In many complex projects, instead of having stakeholders and team members communicate, specifications are tossed over the wall to the downstream activities. Alternatively, “communication” consists solely of signing or declining change requests using a requirements management tool. But let us think aloud whether an agile process could minimize or solve the communication problem. Contractual constraints, such as the need for fixed-price projects and certain types of contracts, shall be ignored at this point. I am looking for a solution for the knowledge transfer problem.

## THE COMMUNICATION TRAP

When it comes to technical systems, there is often a large group of stakeholders who have knowledge about different aspects of the system and what it is supposed to do. A significant problem in systems development — perhaps the main problem — is the transfer of information between the stakeholders and the developers. The Standish Group’s annual CHAOS Reports find the following severe factors for project failure every year:

- > Insufficient information from the customers and users
- > Poor quality of requirements and specifications
- > Changing requirements and specifications

From this we may deduce that systems analysis — the study and communication of the system — is the crucial activity in systems development. The client’s task, in collaboration with the developers, is to depict the full complexity of a system by means of an *exhaustive* set of *high-quality* requirements and to ensure that all the stakeholders understand and agree on the requirements. Nowadays it is clear that these requirements do not have to be conveyed as written specification documents. It is also generally accepted that it is easier not to specify all the requirements perfectly before proceeding, but to elicit and convey them little by little. The important fact is that they *do* have to be conveyed.

## Words Are Without Inherent Meaning

Why is it that we have such a hard time expressing what a system has to do? In order to answer this question, we must look at the way people transfer knowledge between each other.

Our language system is subject to a socially evolved filter. When software engineers talk to each other, for example, they know what a radio button or a drop-down menu is. The same counts for clients and their field of expertise. For example, air traffic controllers know what an SSR code and an abeam position are. Now, imagine if an air traffic controller wrote a specification for a new air traffic control system and handed it to the software developers. The two worlds would collide. Words that are everyday language for the air traffic controller are gibberish to the software engineer. Similarly, the software engineer’s vocabulary is probably incomprehensible to the air traffic controller.

The reason for this is that no common reference model exists, and thus the words are initially meaningless. Only when people draw from common experiences can they assign a unique and unambiguous meaning to an object. Linguists call the act during which a term obtains a meaning a *deictic act*. This act must occur for words to obtain the same meaning for everybody involved in the project.

In projects where there is close collaboration between the client (or domain experts) and developers, these gaps in understanding can be overcome rather quickly due to the closeness of the participants

and their arrival at a common understanding of the work. Experiencing and discussing things together is, and always will be, the surest and fastest way of building a common linguistic reference model. Agile processes, with the closeness of their collaboration, clearly have an advantage here. The participants can create their common linguistic universe fairly easily and then use it to facilitate knowledge exchange.

The problem is that this linguistic universe is created only among those who are involved in the agile project. Sadly, we often experience projects with only a few onsite customers. The entire transfer of knowledge in and out of the project falls on their shoulders. As is common in agile projects, there are few specifications to document the system in a distributable manner. This means the onsite customers are often the translators between the project and the other stakeholders. Often they cannot fulfill this time-consuming and demanding task because of their many other project responsibilities. As I've noted, the project team usually achieves a good level of communication and a common language rather easily, and thus can quickly agree on the requirements for the system. The rest of the stakeholders, however, become lost during this process and find they are no longer part of the communication and coordination chain. This means there is the danger of developing a system that is only to the liking of the onsite customer(s). The other stakeholders, who now feel excluded from the development, may refuse to work with the new product or ask for extensive modifications. When the consolidation of language and opinions does not happen during the analysis phase, it must happen during the rollout phase — the most expensive time for requirements consolidation. Agile projects, and in particular onsite customers, must face the challenge of consolidating language and opinions with the all-important stakeholders early in the project. This is a challenge not to be taken lightly.

## Transferring Knowledge

Let us assume that all important and appropriate stakeholders are involved, they have been integrated into the project, and their opinions have been consolidated. Now, let us look at how knowledge is transferred between humans.

Two principles rule humankind's perception and communication: focusing and simplification (see Figure 2). We humans stop most of reality's sensations before they become part of our memory, or if they do, these sensations are remembered only in a very abstracted form. Human brains are capable of determining what is important and to be remembered and what is less important and can be ignored or immediately forgotten.

Communication is the linguistic expression of our knowledge. It is, of necessity, simplifying. For example, an author assumes that the reader has certain previous knowledge. Otherwise it would hardly be possible to communicate efficiently.

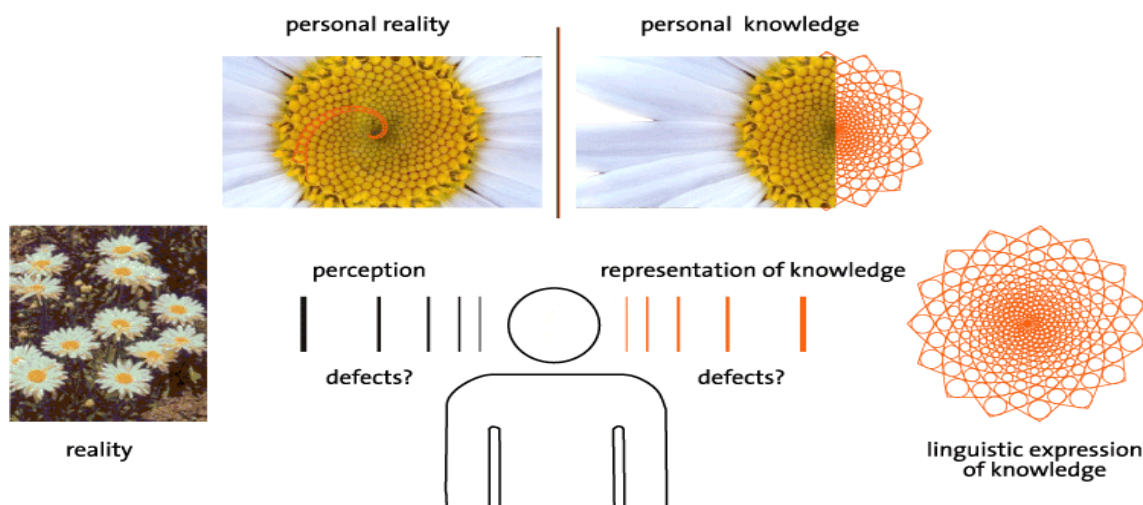


Figure 2 — Reality, personal knowledge, and linguistic expression

While simplification in linguistic expression is needed, it can cause problems when it comes to requirements. However, the linguistic blurs can be systematically remedied to a certain extent, as we will see in the following example.

## Linguistic Nominalizations

Most communications, and especially requirements specifications, are full of nominalizations. A nominalization is a verb or a predicate that has been turned into a noun. They reduce a complex procedure to a single word, and consequently important information about the process is lost. For example:

After a system crash an automatic restart shall occur.

The processes behind the nominalizations *restart* and *system crash* are: “The system crashes” and “The system gets restarted.” Just looking at the latter phrase, the following questions come to the mind:

- > What gets restarted?
- > How is the restart done?
- > With which data is the restart done?
- > Who initiates the restart? (Is it automatic or manual?)
- > What terminates the restart?
- > What happens during restart? (What might abort the restart? What error constraints exist?)

Nominalizations are often found in domains that, over time, have developed an esoteric technical terminology. Consider the typical terms from your own field of expertise. Many of them will be nominalizations.

Generally speaking there is no reason not to use a nominalized term for a complex process, as long as the process is unambiguous or clearly defined. If it is not, however, everyone involved in the conversation will fill this empty word shell with their own imagination — which, as we know from bitter experience, leads to dramatically negative results.

Nominalizations are just one of many known linguistic effects that show up in our communication and specifications. Other problematic linguistic effects are incompletely defined process words, incomplete comparisons, implicit assumptions, and so on. You can imagine why a ragtag team drawn from different companies and different fields of expertise would have a hard time synchronizing the knowledge needed to develop a complex system. The difference between such a team and one that has been working in the same field for a number of years is significant.

## Expertise vs. Ease of Communication

When it comes to systems development, it does not matter whether a misunderstanding due to a nominalization happened in a written specification in a waterfall project or in face-to-face discussion in an agile project. In either case, pointless effort is expended implementing the wrong solution. One solution is to engage a good systems analyst, whose education should enable her to recognize and question all possible linguistic effects during a conversation. A good systems analyst’s education takes years and usually encompasses training in interview techniques, therapeutic concepts, and linguistics. An agile developer, while well versed in architecture and design, writing code, and software testing, will likely not possess these qualifications.

This leads us back to the discussion about specialized expertise versus ease of communication. Is it better to choose project team members who focus their expertise on a single complex field, even though this approach will doubtless require more communication interfaces? Or is it more appropriate to staff your team with jacks-of-all-trades, each of whom knows a little of everything, because this reduces the need for communication interfaces?

There is no right answer, as it depends on the context. Let me illustrate this by using a medical analogy. If ever I need complicated heart surgery, I will insist on well-trained specialists. I would like to have every single step of the surgery performed by someone who has done it successfully 100 before. I do not care if there are 20 doctors standing around the surgical table, each of whom contributes only a tiny

bit to the success of my operation. But I do want to know that the interfaces between the doctors are infallible.

On the other hand, should I fall off my bike during my next mountain bike tour and scrape my knee, I will be happy to be cared for by my husband, as he will be the only available source of medical help. He can do the anaesthesia (a good swig of apple juice and a candy bar to calm my nerves), the disinfecting, the surgery, and the wound care, as well as the psychological support before, during, and after the surgery. There are no interface problems here, and I am able to survive, albeit with a somewhat unprofessional result.

Again, there is no perfect, problem-free solution. Either your project potentially suffers from the interface problem between perfect specialists, or you have no interface problem and everything is in the hands of a semi-professional all-rounders. You are free to choose which problem you'd rather have.

## THE CUSTOMER'S REALITY: BOILING IT ALL DOWN

Now that we have recognized the linguistic problems, I would like to question the stakeholders' grasp of reality. It is very rare that a new systems development starts out in the open countryside. Usually there is an existing system, which has often been in service for many years. The stakeholders' perception of reality is distorted by the characteristics of the legacy system. Rarely do you find stakeholders who are specialists of the internal workings of the legacy system. Most are people who simply use the system.

If, for example, the legacy system protected access by means of a password, the stakeholders are most likely to request a password prompt for the new system. They speak in solutions and forget the underlying essential problem. It is the analyst's task to make these essential requirements visible. Hidden behind the request for a password prompt might be an access protection, a complex role system, a means of time recording, or something entirely different. It might even be that this feature has become pointless, as the building that houses the new system now has an access control system of its own. One of the most important challenges in systems analysis is boiling the stakeholders' perceptions and statements down to their essential core. If this step is left out, many of the legacy system's outdated technological decisions could end up in the new system — this is known as IT folklore.

Typically the process of extracting the essence of the requirements is done by someone who can focus on the requirements and not solutions. This person is usually inappropriate for designing the elegant solution, as we are talking about two mutually exclusive skills. This suggests separation of analysis (working out the essential) and architecture/design/realization (pragmatic decision making). Agile project teams usually have good solution skills. They can create a presentable solution extremely quickly, and so they tend also to think in terms of solutions. This usually means they skip the nerve-wracking task of really getting to the essence of the problem. In my opinion, even in agile projects, it is of great help to have one person who concentrates only on analysis and has nothing to do with finding and implementing solutions.

## WHAT AN AGILE PROJECT CAN LEARN FROM A TRADITIONAL PROJECT

In order to prevent premature thinking in terms of solutions (often from the very start of the project), agile projects should have at least one team member whose task is to keep his head free of solutions. The best person for this job is a coworker who is interested in business and work processes and has little or no experience (or interest) in architecture, design, or implementation. The crucial task of this person is to ensure that during the requirement elicitation process, the team members' minds are set on the essentials, not on the technology.

The specialization of roles in traditional projects can teach an agile project all kinds of methodologies that — at least in the more professional ones — are standard in traditional projects. Methods and notations should not simply be adopted without questioning them, but they should be tested for their value in agile projects and, if necessary, adapted accordingly. For example, knowledge about linguistic effects and methodologies of process modeling can be very useful if there are intense stakeholder interviews during the project. However, adopting all the documentation found in traditional projects usually does

not make sense in agile projects.

In order to avoid trouble during systems rollout, agile project teams — just like traditional project teams — must take into account who their stakeholders are. A responsible selection of stakeholder representatives must be made for requirement elicitation. Relying on just one onsite customer usually means an overly narrow selection of opinions. Do me a favor: say goodbye to the illusion that the process of consolidating different opinions can be avoided by simply building a system and expecting the stakeholders to cope with it and get used to the system over time.

## WHAT A TRADITIONAL PROJECT CAN LEARN FROM AN AGILE PROJECT

Talking with each other is often more effective than writing against each other. In some project situations, direct oral communication has a clear advantage over written communication. Of course, this is even better if the people involved are actually sitting in the same room so their perception of their partners' communication is not limited to the spoken word but can take nonverbal signals into account as well. Things that are still unclear can be questioned immediately — understanding or not understanding may literally be written on your communication partner's face. You can touch pieces of work, documents, and examples; you can show interesting parts; and each party can illustrate their understanding of the situation on the same whiteboard.

Another big advantage of agile projects is that they give the stakeholder a way of clarifying her wishes early. As a usable system exists rather early in the project, the stakeholder can see what she has specified so far. She also gets to see what she left out because, to her, it was so familiar that she implied it rather than explicitly asked for it. Looking at the developing system makes clear whether the developer understood the client who posed the requirements — in other words, whether they both speak the same language.

Early versions of a system also make the project's progress visible, and therefore are good for the project team's morale. Traditional projects produce only paper in early stages of the project. In longer-duration projects this makes it harder to keep the stakeholders motivated.

## CONCLUSION

Are systems engineers losers when it comes to communication? Yes and no — no more and no less than any of us who have to convey complex facts by means of natural language. What counts is the realization that system developers must take a closer look at knowledge transfer and linguistics. Otherwise they will pay for problems that arise when they or their clients ignore communication.

## REFERENCE

1. Hruschka, Peter, and Chris Rupp. *Agile Softwareentwicklung für Embedded und Real-Time Systems mit der UML*. HanserVerlag, 2002.

*Chris Rupp is founder of NLP-based Requirements Engineering. Building on that, she has developed and published pattern-oriented approaches to development. She is the author of numerous international publications and is active as a coach and consultant. Ms. Rupp is the founder and general manager of SOPHIST GmbH. Her method of working covers, among other things, natural linguistics and object-oriented methods, as well as areas of organizational psychology and neuro-linguistic programming (NLP). Personal preferences: people, philosophy, red wine, travel, and the search for the meaning of life. Ms. Rupp can be reached at Tel: +49 40 900 0; E-mail: [chris.rupp@sophist.de](mailto:chris.rupp@sophist.de).*

## FOOTNOTE

<sup>1</sup>Toll Collect was a project to build a system for collecting highway tolls in Germany. The rollout was due 31 August 2003; however, due to development problems, the system has been running with full functionality only since 1 January 2006.