IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4388

A Teleological Approach to Information Systems Development

John M. Artz, PhD.
Associate Professor of Management Science
The George Washington University
Washington, DC 20052, Email: jartz@gwu.edu

THE DIALOG OF GORGIAS

In this, very revealing, Platonic dialog, Socrates faces the sophist Gorgias who is a well-respected public speaker and teacher of rhetoric. Often Gorgias will give a speech in a public place and the Athenians will shower him with praise and money for his efforts. Socrates does not accept money for his teachings, because his only goal is the pursuit of truth, and the scene is set for a showdown between the two. Socrates begins by asking Gorgias who he is. Gorgias responds that he is a teacher of persuasion. The conflict in the dialog is that Socrates is a seeker of truth. He wishes to understand the true nature of concepts such as justice and virtue. Gorgias is a teacher of rhetoric. He believes that there is no truth - that you can convince anybody of anything. Socrates asks Gorgias to give him an example in which persuasion has value. Gorgias says that his brother is a physician and often has to convince his patients to undergo unpleasant treatments for the sake of their health. Socrates asks Gorgias, "What do you persuade the patient to do?" Gorgias responds "Whatever my brother wants the patient to do." There is a moment of realization as Gorgias sees that his art is all technique. While Gorgias is a master of rhetoric and can convince anyone of anything, he is lacking in the knowledge of what he should be persuading people of. He is diminished next to Socrates who is attempting to find out how things should be. While the rhetorical techniques of Gorgias are dazzling, they get him nowhere unless somebody else tells him what to do.

TELOS VERSUS TECHNE

This dialog emphasizes the an important distinction between telos (how things should be) and techne or technique (how to make things that way). It is exactly the problem that we have with technology today. There are many Gorgian software developers who can dazzle you with technique. They can make multicolored objects dance around on the screen. They can recite version numbers and acronyms. They can talk about capacities and capabilities or object models and plug-ins. But when you say "What is the point of what you are trying to do?" they balk just as the sophists balked at Socrates. I do not mean to condemn modern technology nor the people who have worked very hard to master it. I merely want to point out that technology is just embodied technique. And if you cannot answer the question - What should you be doing ? - technique and technology are of quite limited value.

This conflict between telos and techne can be further expanded into a comparison of teleology and technology. Going back to the Greek roots, teleology can be thought of as the study of purposes, while technology can be thought of as the study of techniques or means to achieve ends. Clearly, means to achieve ends rely on ends to be achieved in order to have value and consequently technology without teleology doesn't get you very far. The purpose of this paper is to suggest a teleological approach to information systems development, which changes the focus from the things we could do to the things we should do.

TELEOLOGY

Teleology goes all the way back to Aristotle who was easily Plato's most famous student. Aristotle believed that an adequate understanding of a phenomenon required an understanding of four causes: formal, material, efficient, and final. The formal cause is the shape that a thing takes on. A boat and a picnic table, for example, can be made out of the same material, but take on very different shapes. The material cause is the stuff out of which the thing is made. A boat may be made out of wood or steel. The efficient cause is the procedure by which the thing is made. A boat can be manufactured, constructed from lumber or carved out of a log. The final cause is the ultimate purpose of the thing. The purpose of a boat is to float on water. It is easy to see that a ship builder who does not understand the final cause of a boat may not be very successful in building them.

The final cause is ultimately a teleological explanation, which Aristotle required of all scientific explanations including inanimate or physical phenomenon. This led to some problematic interpretations such as – fire rises because it wishes to return to the sun, or objects fall because they wish to return to the center of the earth. This attribution of purpose to physical objects does not sit well with one's modern sense and indeed Galileo dismissed teleological explanations from his view of astronomy leading eventually to the modern view that physical objects simply follow the laws of nature and do not have any ultimate purpose.

It is fairly easy and appropriate to dismiss teleology from physics, but that does not necessary dismiss this perspective from all scientific and intellectual endeavors. For example, in biology, there is an ongoing debate regarding teleological explanations of biological systems. Can one fully understand the functioning of the kidney if one does not know that the purpose of the kidney is to remove waste materials from the bloodstream? It would seem that one could not. However, does that purpose exist in nature or does the observer ascribe that purpose in order to improve our understanding of the functioning of the kidney. Clearly, this debate can be easily distracted by the philosophical question of whether or

This paper appears in *Managing Information Technology in a Global Economy*, the proceedings of the Information Resources Management Association International Conference.

not purposes exist in nature or if purposes are simply superimposed on nature in order to make things easier to understand. I would gladly accept the second and lessor of these claims by saying that it doesn't matter if purposes exist in nature. Superimposing purposes on nature in order to make things easier to understand is a good enough justification for a teleological perspective.

As we move from biological systems to human systems such as government or economic systems teleological perspectives become increasingly more important. These systems evolved or were constructed to satisfy some sort of human need. To discuss them without discussing the purpose that they serve is to miss the point completely.

The extreme of teleological thinking can be found in engineering where objectives are clearly defined and solutions are constructed to satisfy those objectives. Although it might be theoretically possible to describe a construct such as a bridge over a river in purely physical terms, you probably would not want to drive over a bridge that was built by someone who did not know that the purpose of the bridge was to get cars safely from one side of the river to the other.

Systems engineering was originally very teleological in its approach. Going back to the late 1960's, A.D. Hall provides a description of the systems engineering process, which begins with a problem definition and development of objectives. Originally, systems analysis techniques followed this systems engineering model.But somewhere along the line information systems development lost its teleological focus. What happened?

An important clue can be found in Peter Checkland's Soft Systems Methodology (SSM). Checkland distinguished between "Hard Systems" and "Soft Systems" in that hard systems are fairly well understood problems that are approached from a systems engineering perspective while soft systems are poorly understood problems that must be approached from a learning systems perspective. On the hard systems approach Checkland remarks,

> "Engineering thinking is teleological; it asks: what is the purpose served by the object or system? The engineer works back from the purpose, or objective, and creates an object or system which will achieve that objective." [pg. 274]

By comparison soft systems require a different approach,

> "But in many – perhaps most – managerial problems, at any level, the questions: What are the objectives? What are we trying to achieve? are themselves part of the problem." [pg. 275]

Finally,

> "Whereas systems engineering methodology is a system concerned with achieving objectives, SSM is a learning system." [pg. 278]

Hard systems thinking is essentially teleological in nature and solutions can be defined based on the problem they are intended to solve. Soft systems' thinking allows the development process to be as much a learning process as a software development process. While Checkland's paper did not cause the turning away from teleological approaches, he certainly reflects sentiments that were widely held in the industry at the time. It was difficult to talk about purposes when you had little idea what needed to be done.

The distinction between problem oriented software development and learning oriented software development is an important one and bears future exploration. If a software development team truly has no idea what it is attempting to do, then they need to figure it out somehow. Sometimes that learning occurs by simply diving into the development process. If this is the case then it is important to acknowledge that it is a learning process otherwise one might assume that the resulting system is a solution to the problem rather than a by-product of the learning process. However, if the team does know what it is trying to achieve then that should be articulated as early in the process as possible and a consensus should be reached on the objectives of the solution before any software is developed. Learning oriented software development is extremely inefficient and unproductive in producing solutions and should only be employed when the developers truly do not know what they are trying to achieve.

So the question is reduced to – how often are developers truly learning and how often do developers have a fairly good idea of what they are trying to achieve? I think the answer is – developers probably know a great deal more about what they are trying to achieve than they realize, but the nature of information systems development forces us to take a learning systems approach. If we treat every problem as a unique problem, then every solution will be a unique solution. However, the growing use of application generators and object models, suggests that we have learned a lot in terms of reusing software. What we are not reusing is our experience in problem solving.

A TELEOLOGICAL APPROACH TO INFORMATION SYSTEMS DEVELOPMENT

Information systems development should begin with a problem to be solved. That problem should be articulated and agreed upon. As the process continues the problem statement may have to be refined as developers learn more about using a problem solving approach. This sounds like a learning system and indeed it is. But the difference is that the purpose is to eventually get out of learning system mode. When the project is finished, the team should reflect on the process and figure out what worked and what didn't. Suggestions for improvements in the process should be incorporated into the next development effort. Eventually, the developers will get better at applying a problem solving approach until it is no longer a learning systems process.

The problem should be decomposed into objectives, which are really just subproblems. These subproblems should be articulated and agreed upon just as the problem statement was. And just as was the case with the problem statement, the developers should reflect on the objectives when the project is finished, and then figure out how to improve their articulation of objectives.

The process of stating a problem and then decomposing it into objectives is really just another decomposition technique. Structured analysis decomposes a system into processes. Information modeling decomposes a system into entities. And object oriented analysis decomposes a system into reusable components. Why is decomposition into objectives any better?

There are several answers to this question. First, decomposition into objectives is a higher level abstraction and thus more general. Once the purpose of the solution system is understood, parts of the system may be analyzed further using any of the above techniques. However, the above techniques are all more specific and consequently don't serve to organize the overall purpose of the system being developed. Second, objectives are easier for people to understand. The average person can grasp the concept of what the developers are trying to achieve much more easily than they can grasp the implications of a data flow diagram or object model. Finally, and because objectives are easier to understand, it is easier to gain consensus and develop a shared vision of what the system under development is suppose to do when it is complete. This shared vision is critically important because thou-

sands of design decisions, large and small, will be made at various points by developers based on their understanding of the proposed system. If they do not have a shared vision, the resulting system is likely to be a hodgepodge of conflicting design decisions. Finally, it is much less expensive to correct design flaws earlier in the development process. If there is no agreement on what the proposed system is supposed to do, it is better to find that out when working on the problem statement than during development.

If a problem-oriented approach is so beneficial, why doesn't everybody do things this way? The answer is that defining a problem and decomposing it into objectives is difficult to do correctly. It is difficult for three reasons. First, objectives are less concrete than processes, entities, or reusable components. However, as systems analysis techniques have evolved from flowcharts and decision tables to data flows and entities, the progress has been toward higher levels of abstraction in order to control the complexity of large systems. So decomposition into objectives, although difficult, is necessary as we develop increasingly more complex information systems. Second, it takes a fair amount of experience in defining objectives before one gets very good at it. Approaching information systems development as a learning experience rather that from a process improvement perspective does not allow you to get any better. In order to overcome this problem, developers should approach every system from a problem solving perspective and when they are done, they should reflect on how well they did and what they can do better the next time. Finally, problems and objectives are amorphous terms to most people. In order to be successful in a problem solving approach we need more structure in the concept and a better understanding of problems and objectives. The remainder of the paper will address this issue.

STATING THE PROBLEM

Some times it is difficult to succinctly articulate the ultimate purpose of an information system. Usually there are multiple purposes and multiple levels. And there are many things that need to be done that are not contributing to an ultimate purpose in any obvious way. Information system requirements are usually very difficult to understand and comprehend in their entirety for just this reason. Since developers cannot reliably build a system that they cannot understand, it makes sense to reduce this complexity to something that they can understand. On this issue, Gause and Weinberg offer the following advice: "A possible solution is to regard every design project as an attempt to solve some problem."

Some developers are skeptical of viewing software development as solving a problem. Frequently, software is developed to exploit an opportunity. However, the difficulty here lies in the common semantics of the word 'problem' versus a more precise technical definition. A problem is not necessarily something 'wrong' with the organization. Again, Gause and Weinberg offer some direction: "A problem can be defined as a difference between things as perceived and things as desired." A problem here is a gap between the way things are and the way we would like things to be. Since nobody's life or organization is perfect there are many instances of a gap between the current and the desired state. So every person and every organization has problems that do not necessarily reflect negatively upon them.

But not all problems are solvable problems. If a person thinks that the desired state for their organization is that everybody should be a millionaire then there is a gap that is unlikely to be closed. However, if a person thinks that paychecks should come out on time and be accurate, then it is a gap that it is possible to close. This idea is then embodied in the concept of a 'solvable problem.' A solvable problem is a problem in which the desired state can be achieved given the resources available to the problem solver. That is, there is a gap between the current and desired state. It is a gap that we know how to close given the necessary talent and other resources. And those resources are available.

Now we must take a fairly large step and claim that in a teleological approach to information systems development we should only work on solvable problems. This claim must raise eyebrows and it certainly flies in the face of the Soft Systems Methodology. SSM claims that we often do not know what we are doing in information systems development for a wide variety of very good reasons. How then would we ever build a new system or develop software that has never been developed before.

There are two important responses. First, I believe that we grossly exaggerate the extent to which software being developed is truly new. It is much more likely that developers who are unaware of what other developers have done are simply solving the same problem again in a different way. In recent years we have seen a proliferation of object models which suggest that there are common solutions to information systems problems and that it makes much more sense to learn about and utilize common solutions rather than constantly reinventing everything.

Second, if the software to be developed is truly new, then it is a learning process not a software development process. The problem to be solved is the gap between the developers current understanding of the application or technology and the level of understanding that would be required to approach the application from a teleological perspective. Once that problem is solved, the developers can focus on the application problem. Certainly no one would ever board an airplane that was built so that the engineers could learn about aerodynamics. But we seem to think that it is O.K. for an organization to rely on an information system that was built so that the developers could learn about the application.

It is certainly fair to observe that information systems typically solve many problems. This may very well be true. But it is also fair to observe that some problems are more important the other problems and some problems are subsumed in larger problems. We cannot loose cite of the fact that our analytical techniques in information systems are driven by limitations in human cognition. Decomposition techniques, for example, are a result of a recognition that we cannot understand complexity. We have to break it down into pieces and then understand the pieces. In the same way, a multifaceted problem may be a more accurate description of the situation, but if nobody can understand it, then it serves little purpose in the development process. A clearly stated problem may necessarily exclude something that is otherwise deemed important. But it may also increase the likelihood of success in the development process.

DEFINING OBJECTIVES

The weakness in current methods of decomposition such as data flows, entities or objects is that they do not address what the system under construction is supposed to do. They may address the way things are currently done, or the way they could be done, but they do not address the way they should be done in order to solve the problem at hand. Using objectives in the system development process provides a means for addressing the purpose of the system in a highly structured and systematic fashion, thus filling the gaps left by traditional methodologies. Historically, the problem with using objectives has been the lack of structure in the process of defining objectives. Objectives are a decomposition of system purpose in the same sense that modules are a decomposition of system function. Viewing objectives in this way makes it possible to distinguish well stated objectives from poorly stated

objectives. Further it is possible to identify different kinds of objectives (component, competing, and constraining) and how they relate to each other.

An objective is a subproblem that arises when a problem space is decomposed into (relatively) independent components. This is to say that an objective is a solvable subproblem. An objective is a definition of purposeful activity that not only defines a desired end state but also carries within its statement some understanding of how that end state will or should be achieved. Viewed as a decomposition of the problem space, objectives should have the following characteristics:

1) Objectives should be mutually independent, and each objective should solve one aspect of the problem in its entirety.

2) Objectives should be collectively comprehensive so that if all objectives are met the problem will be solved.

3) Objectives should lend themselves to further hierarchical decomposition so that if an objective is too large to tackle directly, it can be broken into subcomponents that again have the characteristics being stated here.

4) Every objective should state some important aspect of the resulting solution and by implication should indicate something that is not important. Hence, objectives of this type should guide the developers solving the problem at every stage as to the goals of the solution they are trying to produce.

In addition, the following aspects of objectives should also be observed:

1) There are three levels of objectives and objectives must match the level of the problem that they are addressing.

2) There are as many different kinds of objectives as there are interests in the solution.

3) Objectives can be related to each other as components, competitors, or constraints. As components they can be organized hierarchically as a means ends analysis of the problem space. As competitors they represent conflicting goals and must be examined in a trade-off analysis. As constraints they place limitations on the possible solutions to the problem under consideration.

4) Some objectives are units of work toward which action will be directed. These actions will be human and organizational actions. Hence, these objectives must be achievable units of work.

For a more detailed discussion of objectives and their role in information systems development see Artz [1996].

SUMMARY AND CONCLUSIONS

For well-understood problems, a teleological approach is the most effective because it focuses resources towards the solution of the problem. The problem should be a fairly well-understood and well-defined solvable problem. This problem can then be further decomposed into objectives, which are essentially solvable subproblems. This raises the question – how often do we address well-understood and well-defined solvable problems in information systems development. It seems that we might do this more often than we currently do, by employing a problem solving approach and improving it over repeated development efforts.

REFERENCES

Artz, John (1996) Information Systems Development By Objectives. Proceedings of the 1996 Eastern Acadamy of Management Meeting "New Connections in the Information Age" Crystal City, Virginia. March 1996.

Beckner, M. (1967) Teleology. In Edwards, P. (ed.) *Encyclopedia of Philosophy*. Vol. 8, 1967, pp. 88-91.

Checkland, P.B. (1989) Soft Systems Methodology. Human Systems Management. 8(4). Pp. 271-289.

Gause, D. and Weinberg, G. (1989) Exploring Requirements: Quality Before Design. Dorset House Publishing.

Hall, A.D. (1969) A three dimensional morphology of systems engineering. IEEE Transactions on Systems Science and Cybernetics. Vol. SSC-5., No. 2., pp. 156-160.

Losee, J. (1993) A Historical Introduction to the Philosophy of Science. (3e) Oxford University Press.

Nagel, Ernest (1979) Teleology Revisited and Other Essays in the Philosophy and History of Science. Columbia University Press.

Plato. Gorgias. Translated by James H. Nichols, Jr. 1998.

Sugrue, M. (1998) Plato, Socrates, and the Dialogues. The Teaching Company. Superstar Lecture Series.