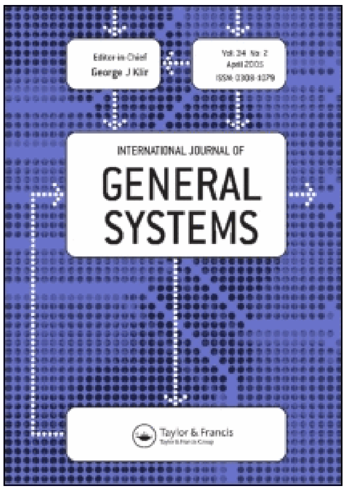

International Journal of General Systems
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713642931

Valid models require defined levels

A. T. Bahill [a]; F. Szidarovszky [a]; R. Botta [b]; E. D. Smith [a]
[a] Systems and Industrial Engineering, University of Arizona, Tucson, AZ, USA [b] BAE Systems, San Diego, CA, USA

First Published:October2008

To cite this Article Bahill, A. T., Szidarovszky, F., Botta, R. and Smith, E. D.(2008)'Valid models require defined levels',International Journal of General Systems,37:5,553 — 571

To link to this Article: DOI: 10.1080/03081070701395807

URL: http://dx.doi.org/10.1080/03081070701395807

International Journal of General Systems
Vol. 37, No. 5, October 2008, 553–571

Taylor & Francis
Taylor & Francis Group

Valid models require defined levels

A.T. Bahill[a]*, F. Szidarovszky[a], R. Botta[b] and E.D. Smith[a]

[a]*Systems and Industrial Engineering, University of Arizona, Tucson, AZ, USA;* [b]*BAE Systems, San Diego, CA, USA*

(*Received 9 January 2006; final version received 8 November 2006*)

A common mistake in modeling systems is mixing elements of different levels in the same model. This paper explains levels, shows how levels can be obtained with decomposition techniques and gives many examples of levels in models, architectures and frameworks. Examples are given that show the confusion that results when levels are mixed. The paper shows that complex models are usually composed of many aspects. Each aspect has levels and the levels of one aspect are different from the levels of another aspect. The model must be constructed using elements of the same level for each aspect.

Keywords: modeling; abstraction; decomposition; hierarchy; systems science

1. Examples of levels in different disciplines

Identifying levels in models is a basic modeling principle. This principle is not specific to any field of application: it applies to models in general. This paper shows many examples of levels in many fields of endeavour. The concepts presented in this paper are not restricted to engineering.

A common mistake in modeling systems is mixing elements of different levels in the same model. Therefore, an important task in designing and modeling systems is identifying the level of the proposed model and its elements. However, the term level is often used rather cavalierly, so here we will give a definition, and several examples. First the dictionary definition: "level n. 1.a. Relative position or rank on a scale. 1.b. A relative degree, as of achievement, intensity, or concentration… 3. Position along a vertical axis; height or depth."

The concept of examining a system at many physical orders of magnitude was presented by Boeke (1957). Later Eames and Eames (1968) created a film of this concept and finally Morrison and Morrison (1977) popularized the idea with the book *Powers of Ten*. These authors treated physical decomposition as more and more levels of detail.

We tried to create a canonical prescription for levels in systems, but the best we could do is present many examples of levels from many different fields. After our presentation of these examples we present half-dozen examples of failures that were caused by level mistakes. Then we present some generalizations that are meant to help modelers and designers to define levels and avoid mistakes in mixing elements of different levels. We also suggest some reasons why it so hard to keep levels consistent in a system. One of the primary reasons is that different aspects of a system have different levels.

* Corresponding author. Email: terry@sie.arizona.edu

ISSN 0308-1079 print/ISSN 1563-5104 online
© 2008 Taylor & Francis
DOI: 10.1080/03081070701395807
http://www.informaworld.com

Physical Decomposition is a commonly used decomposition technique. For systems engineers, physical decomposition often offers these levels (Morganwalp and Sage 2003)

1. Enterprise
2. Family of systems
3. Systems
4. Segment
5. Element
6. Subsystems
7. Components
8. Subassembly
9. Parts

An *army* is divided into divisions, regiments, battalions, companies, platoons and solders. In Julius Caesar's time there was a ten to one ratio of the sizes of these elements. That is, ten solders comprised a platoon and there were ten platoons in a company, which was commanded by a centurion.

Functional Decomposition is another way to create various levels. MIL STD 499A said that the preferred way to design a system was functional decomposition. Start with the top-level system function and decompose it to lower and lower levels. Students used to query, "When do you stop decomposing?" We used to answer, "When you get a function small enough to be designed by a team of people." However, we now answer, "When you get a function that can be implemented by a commercial off the shelf component (Botta *et al.* 2006)." Here is an example of functional decomposition for an automobile:

Move people

- Accommodate people
    - Provide seats
    - Entertain people
        - Provide auditory stimulation
        - Provide visual images
    - Provide comfortable environment
        - Heat interior
        - Cool humans in the car
        - Provide fresh air
        - Sound proof interior from exterior
    - Protect occupants
        - Deploy air bags in a crash
        - Absorb collision energy in crumple zones
- Move the automobile
    - Overcome friction
    - Accelerate
        - Produce lateral acceleration
        - Produce translational acceleration
            - ≫ Increase forward speed
            - ≫ Decrease forward speed
- Move in a smooth manner
    - Minimize shocks

In the Business community, a *work breakdown structure* (WBS) shows different levels of work packages. Project Management books (Kerzner 2003) describe the WBS like this:

Work breakdown structure levels

Level	*Responsibility*	*Example*
1	Management	Program
2		Project
3		Task
4	Technical	Subtask
5		Work package
6		Level of effort

In system design, *use cases* describe the required functionality of a system. Cockburn (2001) says that an important slot in a use case is the level. His levels from top to bottom are

Levels in use cases

Design scope	*Goal level*	*Icon*
Organization (black box)	Very-high summary	Cloud
Organization (white box)	Summary	Kite
System (black box)	User goal	Sea level
System (white box)	Subfunction	Fish
Component	Too low for a use case	Clam

Cockburn (2001) says that you raise the level by asking, "Why is the actor doing this?" and you lower the level by asking, "How is the actor going to do this?" J. D. Baker said, "A use case diagram with the use cases of Login and Play Global Thermonuclear War would be difficult to implement." Login is at the lowest level, commonly called a sub-function or sub-goal. Play Global Thermonuclear War is an example of the highest-level use case, commonly called the Very-high summary level or an Essential use case. Use case diagrams should not mix use cases of different levels.

The National Security Agency (2006) decomposes *signals intelligence* as

Intelligence—applied knowledge
Knowledge—facts relationships, context
Information—discrete facts, entities
Data—bit streams, receptacles
Signals—electrical impulses, sensors

The field of *threat assessment* has the following levels of abstraction:

Situation and threat assessment
Adaptive agents
Template matching
Feature extraction
Database assimilation
Fused multispectral images
Sensor data

The lowest level is data from sources such as synthetic aperture radar, infrared detectors, GPS, electro-optical sensors and intelligence. Data from many sources are fused into multispectral images and these are condensed into a database (Waltz and Llinas 2000). Then particular features are extracted (features include things such as objects, heartbeats of an electrocardiogram, etc.). These features are matched against templates such as wing, wheel, QRS complex, etc. These templates are passed to agents (Ferber 1999). These agents support one or more of the relevant alternatives. These agents can trigger a preplanned sequence of activities. For example, one might recommend that if agent A does X and agent B does Y, then hypothesize Q, for which an appropriate sequence of actions might be to scan agent A with source R, launch airplane S and if source R shows that agent A did Z then launch weapon T.

Science is organized in levels as shown in the following table based on Kang and Mavris (2005).

Science is organized in levels

Level (scientific field)	*Typical mechanisms*
Ecosystem (ecology)	Predation, symbiosis, mimicry
Organism (biology)	Growth factors, apoptosis, morphogenetic operators
Cell (biology)	Mitosis, meiosis, genes
Organelle (microbiology)	Enzymes, membranes, transport
Molecule (chemistry)	Bonds, active sites, mass action
Atoms (physics)	Protons, neutrons, electrons
Nucleus (physics)	Quarks, gluons

The retina of the eye is arranged in layers. First, we will give an anatomical description (structure, what) of these layers and then we will give a physiological description (function, how). This example is included to show that the concept of levels is not restricted to models of man-made systems.

Anatomically the retina has seven layers. The dark stained layers contain cell bodies and the white layers contain axons and dendrites. The photoreceptor layer contains the outer segments of the rods and cones. The outer nuclear layer contains the cell bodies of the photoreceptors. The outer plexiform layer contains the connections from the photoreceptors to the bipolar and horizontal cells and also the horizontal interconnections of the horizontal cells. The inner nuclear layer contains the cell bodies of the bipolar, horizontal and amacrine cells. The inner plexiform layer contains the connections from the bipolar cells to the ganglion cells and the amacrine cells and also the dynamic horizontal interconnections of the amacrine cells. The ganglion cell layer contains the cell bodies of the ganglion cells. And finally, the optic fiber layer contains the axons running from the ganglion cells to the brain (Warwick 1976).

Physiologically each layer of the retina has a different purpose. The vertical signal pathway is from the rod and cone photoreceptors to the bipolar cells to the ganglion cells and then up to the brain. However, there are also horizontal signal processing layers (Kaufman and Alm 2002).

Diffuse light excites a photoreceptor and turns it on. That photoreceptor excites its bipolar cell and that bipolar cell in turn excites its ganglion cell. However, the diffuse light also excites the neighboring photoreceptors. They excite their horizontal cells, which inhibit the central photoreceptor's bipolar cell. Therefore, the central bipolar cell is silent. However, a *small spot* of light excites only the central photoreceptor and not its neighbors. Therefore, the central bipolar cell will not be inhibited by its neighbors. It will get excited and it will excite its ganglion cell, which will fire at a high rate. In summary, photoreceptors respond to individual points of light. Whereas, bipolar cells respond to static patterns of light. In the case described here, they respond to a small spot of light surrounded by darkness.

Next, the bipolar cells excite their ganglion cells and also the surrounding amacrine cells. If the light pattern is moving then the amacrine cell modulation will affect the firing of the ganglion cells. Thus, the ganglion cells respond to dynamic patterns of light. There are many types of ganglion cells. We have just described on-center off-surround cells. There are also on-off cells and opponent color processing cells. In the retina, the photoreceptors respond to illumination. All other cells in the retina respond to patterns, movement and transients. The ganglion cells send their signals to the lateral geniculate nucleus and from there to the visual cortex.

Our visual system sees borders and contours. We see the world as a pattern of lines. This system of lateral inhibition in the retina is the first step towards sharpening contours and picking up on borders between light and dark. The ganglion cell ignores diffuse light, but a specific pattern of light will turn it on. Higher up in the cortex, all these dots will be combined into lines, which will be combined into curves, etc. Other areas of the cortex of the brain have a similar layered structure.

The *Hearsay speech understanding system* used the blackboard architecture (Erman *et al.* 1980). The key features of the blackboard architecture are multiple cooperating sources, multiple competing hypotheses, multiple levels of abstraction, feedback to the sources and the blackboard, which is an associative memory. In order to understand speech, the Hearsay system used the acoustic waveform to identify phonemes, which were then transformed into syllables. The syllables were used to generate word candidates and the words were used to hypothesize phrases. Finally the phrases were arranged into potential sentences. Each of these levels had its own specialized computer processor that communicated with other processors through the blackboard. Figure 1 shows an abstract blackboard with information being passed between levels in a speech understanding task.

Artificial neural networks are hardware and software systems that are very good at pattern recognition, if many examples of correct categorization are available. There are a dozen common types of artificial neural networks. Most of the variation is in how the weights are adjusted. Back propagation is probably one of the most common types. Back propagation artificial neural networks typically have five layers: an input layer, the input-weight layer, the hidden layer, the output weight layer and the output layer. Often a second hidden layer and another layer of weights are added (Szidarovszky and Bahill 1998).

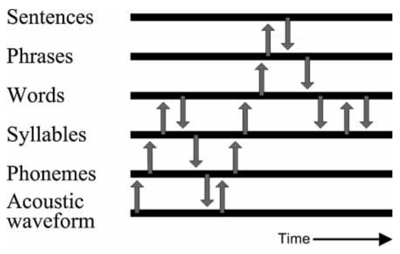

Figure 1. Illustration of a blackboard with information being passed between levels in a speech understanding task.

Levels in *digital computer simulations* are summarized nicely in the following table derived from Zeigler *et al.* (2000).

Digital computer simulations are written in levels

Level	*Name*	*What we know at this level*
5 (highest)	Coupled component systems	Components and how they are coupled together. The components should be systems themselves, thereby producing hierarchical structures
4	State transitions	How states are affected by inputs, that is, given an initial state we know the new state after the input is finished; what outputs are generated in each state
3	System experiments	Given an initial state, every input trajectory produces a predictable output trajectory
2	Input/output behavior	Time-indexed data collected from the source system, which consists of input/output pairs
1 (lowest)	Source system	Inputs to apply, variables to measure and how to observe them over time

They write that systems analysis is a process for modeling and subsequently understanding existing systems. It is a process of going down through the levels. Whereas systems design is a process of climbing up through the levels. This compares well with Cockburn's statement that you raise the level by asking, "Why is the actor doing this?" and you lower the level by asking, "How is the actor going to do this?"

What seems to be common to all of these examples is that (1) there are about seven plus or minus two levels (Miller 1956), (2) each level addresses a different purpose and (3) the low-levels have detailed information and the high levels are abstract.

2. Related terms

Abstraction. So now, let us try to describe abstraction. Here are some dictionary definitions: "abstract adj. 1. Considered apart from concrete existence . . . 4. Thought of or stated without reference to a specific instance." "abstraction *n*. The process of extracting the underlying essence, removing dependence on real world objects." Abstraction allows designers and modelers to describe a problem at a high level by hiding low-level information. In general, the higher the level of abstraction, the more things the system hides from the user.

In 1945, Pablo Picasso made a wash drawing of a bull. A month later, he finished his 11th state of abstraction, which was composed of six lines. Célestin said, "When you stand before his eleventh bull, it's hard to imagine the work that went into it" (Picasso 2006). At another time, he did an abstraction of his wife's face in six stages. Picasso and Henri Matisse concurrently developed abstractionism in modern art.

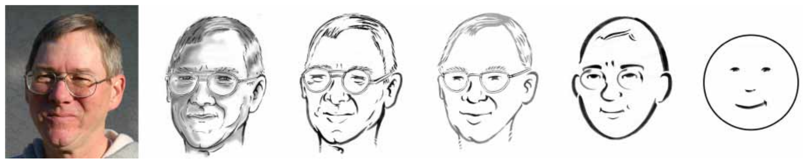

Figure 2. A sequence of images going from real to abstract that describe respectively a single person under certain circumstances, dozens of people, hundreds of people, thousands of people, millions of people and (almost) all people. Drawings are by Angelo Hammond. Copyright ©, 2005, Bahill, from http://www/sie.arizona.edu/sysengr/slides/ used with permission.

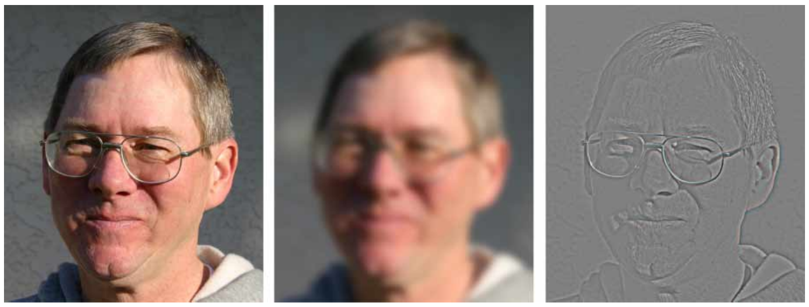
Figure 3. An image of a person (left), a low-pass filtered version of that image (center) and a high-pass filtered version (right). Copyright ©, 2005, Bahill, from http://www/sie.arizona.edu/sysengr/slides/ used with permission.

McCloud (1993) relates abstraction to universality. The more abstract a cartoon image is, the more people it will describe. The photograph in Figure 2 is singular: it describes one person. Whereas the smiley face cartoon is universal: it describes all people.

Abstration is not just eliminating detail; rather abstraction focuses on and changes specific details. Depending upon the details you focus on, you will get different abstractions of the same object. For example in software you might be interested in a sequence of objects {a–f}. If all you want to do is extract or replace objects, then you can abstract this sequence as an array. But if you also want to insert and remove objects, then you should abstract it as a list. Therefore, low-pass or high-pass filtering is not abstraction. Abstraction may require human intelligence, not a digital filter. Figure 3 shows low-pass and high-pass filtering.

Generalization is different from abstraction. Bjarne Stroustrup said, “There is always the temptation to provide just the solution to a particular problem. However, unless we try to generalize and see the problem as an example of a general class of problems, we may miss important parts of the solution to our particular problems and fail to find concepts and general solutions that could help us in the future.”

Generalization usually goes from the bottom to the top. For example, you might start with a pile of data points. You could organize them into a scatter graph and then fit the data with a function (e.g. spline) or statistical technique (e.g. Poison distribution). To further generalize you could generate equations to describe the data. Finally, those equations could be organized into vectors and matrices. Each step up removes detail and increases the generality of the result.

Going from the top to the bottom, there is a design method called *consistent elaboration* (Wymore 1993, pp. 178–180). In it, the designer starts with a high-level system model and then iteratively and systematically expands the input ports, output ports and state variables to create lower levels.

In the unified modeling language (UML) (OMG 2006) generalization, which is related to inheritance, takes you up the hierarchy and specialization takes you down.

CMMI Levels

Level	*Characterization of the maturity of an organization’s processes*
5 Optimizing	Focus is on process improvement
4 Quantitatively Managed	Process is measured and controlled
3 Defined	Process is characterized for the organization and is proactive
2 Managed	Process is characterized for projects and is often reactive
1 Initial	Process is unpredictable, poorly controlled and reactive

The capability maturity model integration (CMMI) categorizes organizations into five levels according to the maturity of their processes (Chrissis *et al.* 2003).

The low-level processes are detailed and concrete. The high-level processes are more abstract. Organizations are supposed to build themselves up, adding levels as they mature.

Granularity refers to the level of detail or the level of abstraction of each model. The analogy is to rocks. Tiny rocks are called sand, small rocks are called gravel, larger rocks are called pebbles, even larger ones are river stones and big ones are called boulders. A driveway is often paved with gravel, where each rock is of the same size.

3. Level of abstraction is usually relative

Abstraction level is relative rather than absolute. It is a partial ordering over elements. We can say that one element is at a higher-level than another is, but we cannot describe that level absolutely.

For example, a government agency might have the following WBS

1. Homeland Security
2. Navy
3. Aircraft carrier task force
4. Airplanes
5. Weapon systems
6. Missiles

Whereas a missile manufacture might have

1. Family of missiles
2. Air-to-air missile
3. Ordnance
4. Safing system
5. Arming, firing and fuzing
6. Thermal batteries

And the battery manufacture might have another WBS . . .

An exception to the relativeness of decomposition is the open system interconnection (OSI) architecture for telecommunication. The OSI uses a seven-layer model where each successive abstract layer is built on top of a lower abstract layer (CISCO 2006).

1. The *application layer* contains user programs, such as Telnet, XML, HTTP, SSH and FTP.
2. The *presentation layer* formats and encrypts data to be sent across a network. Protocols at this layer are part of the operating systems of each computer.
3. The *session layer* specifies how to establish a communication session with a remote computer. It specifies how to login to a remote computer and how to authenticate passwords. It provides a way of knowing where to restart the transmission of data if a connection is temporarily lost.
4. The *transport layer* hides details of network-dependent information from the higher layers. The transport layer sends packets from machine-to-machine and guarantees that packets arrive in the correct order and number. The Transmission Control Protocol (TCP) is primarily in this layer.
5. The *network layer* determines how messages are routed within the network. It transfers data sequences from source to destination. The Internet Protocol (IP) is primarily in this layer.

6. The *data link layer* is concerned with the transmission of blocks (frames) of data. At this layer, data packets are encoded and decoded into bits. It detects and possibly corrects errors that may occur in the physical layer.
7. The *physical layer* transmits a bit stream at the electrical and mechanical level. It provides the hardware for sending and receiving data on a carrier. RS-232 is a standard in this layer.

Entities in one level can communicate with entities in the same level and with entities one layer above and below: but that is it. Communications cannot skip a level. The OSI is only a model, not a standard. Therefore, it is not mandatory. The more modern universal serial bus (USB) Specification (www.usb.org) is an Open Standard. It only considers the lower three layers: Function Layer, USB Device Layer and USB Bus Interface Layer. Likewise the integrated services digital network (ISDN) operates in the lower three layers. Internet Reference Model has only five layers and they encompass all seven OSI layers (Comer 2004).

Software is often designed with the following layered architecture (Evans 2004).

User interface (or presentation) layer
Application layer
Domain layer
Infrastructure layer

Some interlevel interactions such as dependencies and action initiation may be unidirectional. This simplifies the implementation.

4. Why is defining levels important?

Consider the activity diagram of Figure 4 from baseball for one pitch and responses to it. Assume (1) a groundball is hit into fair territory, (2) the ball is fielded and thrown to first base, (3) the fielder on first base catches the throw and (4) there are no other base runners.

Now consider the Batter in this activity diagram. We could model the state of his mind with the following attributes and states:

experience: rookie, veteran, imminent free agent
salary: considered too low, considered too high, commensurate with earned respect
physiology: age, health, on disabled list
competition: other players at his position

Does this model fit with the activity diagram? No, because it is at a different level.

Let us reconsider the Batter in the activity diagram again. We could model the state of his mind with the following attributes and states:

count: balls, strikes, outs
mentalModels: speed of last pitch, umpire's last call
situation: runners on base
signals: last signal from coach

Does this model fit with the activity diagram? Yes, because it is at the same level.

Figure 5 shows class diagrams for the Batter of the activity diagram in Figure 4. Class diagrams have three compartments: the top compartment contains the name, the middle compartment contains the attributes (things that are stored in memory) and the bottom compartment contains the operations or functions that the class performs. Figure 5 shows a class

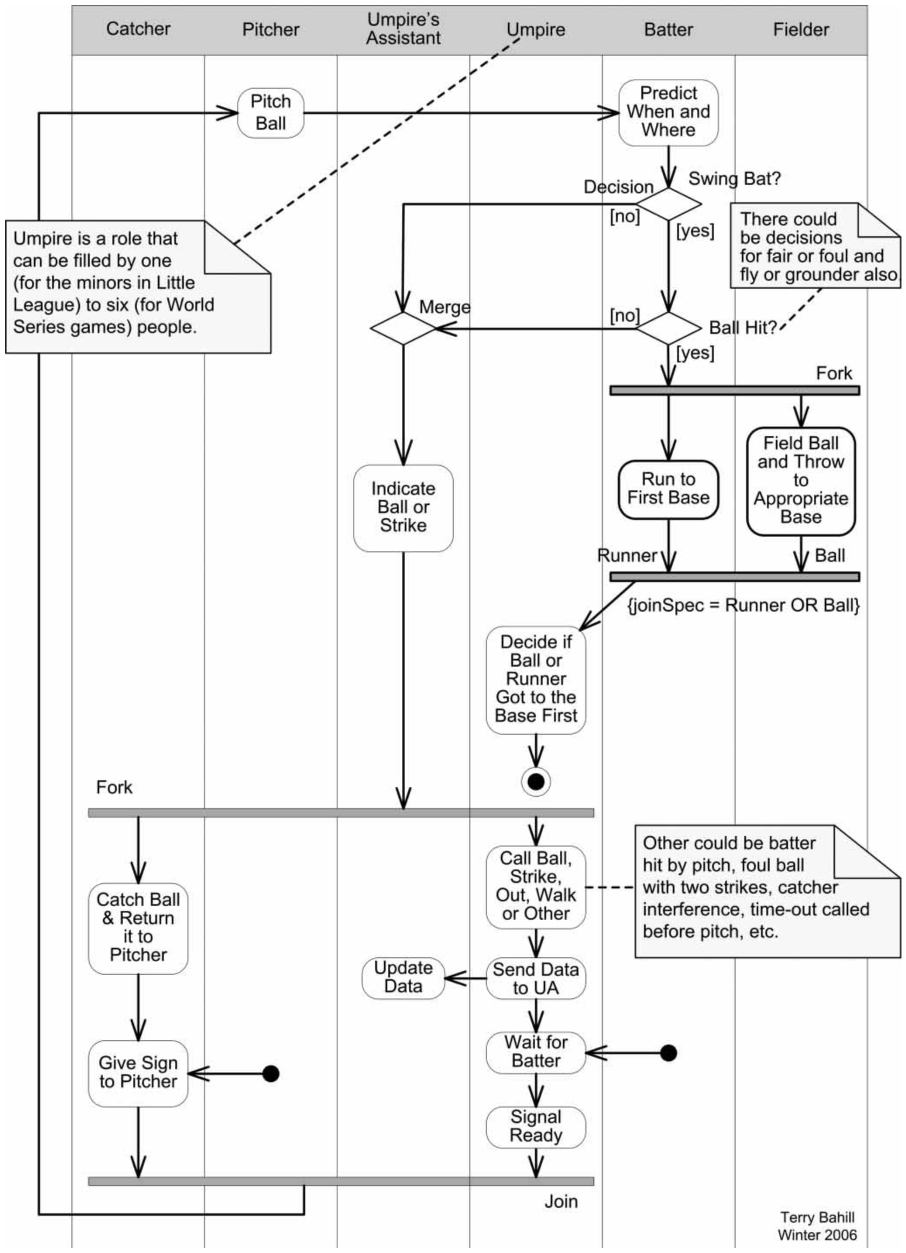
Catcher
Pitcher
Umpire's Assistant
Umpire
Batter
Fielder
Pitch Ball
Predict When and Where
Decision
Swing Bat?
[no]
[yes]
Umpire is a role that can be filled by one (for the minors in Little League) to six (for World Series games) people.
There could be decisions for fair or foul and fly or grounder also
Merge
[no]
Ball Hit?
[yes]
Fork
Field Ball and Throw to Appropriate Base
Run to First Base
Indicate Ball or Strike
Runner
Ball
{joinSpec = Runner OR Ball}
Decide if Ball or Runner Got to the Base First
Fork
Call Ball, Strike, Out, Walk or Other
Other could be batter hit by pitch, foul ball with two strikes, catcher interference, time-out called before pitch, etc.
Catch Ball & Return it to Pitcher
Update Data
Send Data to UA
Wait for Batter
Give Sign to Pitcher
Signal Ready
Join
Terry Bahill
Winter 2006

Figure 4. An activity diagram for one pitch and a partial response to it (Assuming it is a groundball hit into fair territory. The fielder on first base catches the throw. There are no other base runners.). Copyright ©, 2004, Bahill, from http://www/sie.arizona.edu/sysengr/slides/ used with permission.

diagram at the salary negotiation level (left) and a class diagram at the batting level (right). The diagram on the right fits the activity diagram of Figure 4; whereas the one on the left does not. If in a single model one diagram is at one level and another is at a different level, then you should expect a failure to communicate.

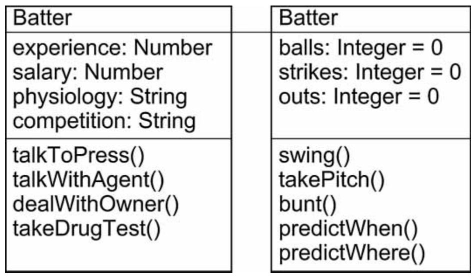

Figure 5. Class diagrams for the Batter of Figure 4 at the incorrect (left) and the correct (right) level.

Bahill has stated, “The most common student mistake in modeling that I have observed in four decades of teaching is creating elements at different levels in the same model; for example writing a use case at a high level and a creating a class diagram at a low level.”

Scaling up an old design can cause levels mistakes. It would be easy to design a red brick arch to span a small creek, but if this same design were used for a bridge across the Grand Canyon, you should expect failure. A design that is successful at one level might not be successful at a higher-level. Bahill and Henderson (2005) studied two dozen famous failures and concluded that two of them were due to faulty scale up of an old design. The next four examples are based on this paper.

The original *Tacoma Narrows Bridge* was a scale up of an old design. But the strait where they built it had strong winds: the bridge became unstable in these crosswinds and it collapsed in 1940. The George Washington Bridge over the Hudson River in New Your City was built with the same design ten years earlier. It came in under budget and ahead of schedule. It currently carries 300,000 vehicles per day. It did not resonate and self-destruct. The George Washington Bridge had a Ratio of Length to Cross Section Area of about 4. For the Tacoma Narrows Bridge

Scale up of an old bridge design

Bridge	*Length (m)*	*Width (m)*	*Depth (m)*	*Ratio of length to cross section area*
George Washington Bridge	1450	37	10	4
Tacoma narrows Bridge	1800	12	2.4	63

they increased the level of the Ratio of Length to Cross Section Area to about 63. This change in level caused the failure. Scaling up an old design often causes level mismatches.

The French *Ariane 4 missile* was successful in launching satellites. However, the French thought that they could make more money if they made this missile larger. So they built the Ariane 5. It blew up on its first launch destroying a billion dollars worth of satellites. The designers faultily assumed that their design that worked at one level of physical size would work at the next higher-level. It did not. The software and the hardware were at different levels. The computer software was at the old level and the newer, bigger Ariane 5 missile was at a higher-level.

The *Space Shuttle Challenger* blew up in 1986. Beforehand, perceptive engineers were focused on the low-level risks: low temperature and possible failure of the O-rings (Feynman 1985, Tufte 1997). But the managers were so focused on the high-level political consequences of delaying the launch that they did not pay attention to low-level details like the effect of temperature on O-rings. One group was thinking a low-level and another group was thinking at a high level.

On the *Mars Climate Orbiter*, the prime contractor, Lockheed Martin, used English units for the satellite thrusters while the operator, JPL, used SI units for the model of the thrusters. Therefore, there was a mismatch between the space-based satellite and the ground-based model. Every time the thrusters fired, error accumulated between the satellite and the model. This caused the calculated orbit altitude at Mars to be wrong. Therefore, instead of orbiting, it entered the atmosphere and burned up. Giving the units of measurement is a low-level modeling task. Using different levels of measurements is a fatal modeling flaw.

The system designer must produce designs for both the *product* and the *process* that will produce it. In one of Bahill's graduate system design classes, the best student project design contained the following requirements (Abadi and Bahill 2003, p. 112)

1. Acquisition Time (in months) baseline = 3 months. The number of months required to complete the design in order to get the product to market.
2. Acquisition Cost (in dollars). The project should be completed within the budget established for this effort: $425,000.
3. Manufacturing Cost (in dollars/racket) Baseline = $24. The tennis racket design must consider the expense of manufacturing.

These three requirements were presented at the same level: this was a mistake. The second of these requirements is clearly a process requirement and therefore should not be in the product documents. They were presented at the same level, implying that they were siblings. In reality there should have been a *trace* relationship, a parent–child relationship, between them. In the UML community (OMG 2006) the process documents are called the Business Model, and they are separate from the product documents. In a requirements specification, tractability is very important (Daniels and Bahill 2004). Traceability is a manifestation of levels.

Possible reasons for level mistakes. A typical system evolves through the following life cycle phases: State the Problem, Discover Requirements, Investigate Alternatives, Design the System, Implementation, Operation and Maintenance and Retirement. Different components will progress through their life cycle phases at different rates, which should produce the opportunity for level mismatches. Other potential reasons for level mismatches include not understanding levels in the initial formulation and organizational hierarchies that create havoc.

Models with use cases at different levels of abstraction will be hard to understand. Furthermore some parts of the system will seem to be high priority just because they are elaborated more, whereas other parts will seem less important because they are modeled briefly (Övergaard and Palmkvist 2005).

Mistakes in mixing elements of different levels are common in abstract systems, but not in physical systems, because such mistakes would be ridiculously obvious in physical systems. For example, a model railroader would never put an N-gauge tunnel in an HO gauge model railroad. Similarly a naval modeler would never put a 1:144 scale model of an F-22 on the deck of a 1:500 scale model of the U. S. S. Ronald Reagan.

5. Multiple aspects

Not only do systems have multiple levels, but there are also multiple aspects that must be considered. In this paper, we have shown functional decomposition, physical decomposition and task decomposition (a WBS). There is also requirements decomposition (which we have not shown in this paper, see Hooks and Farry 2001 and Bahill and Dean 2007). We could look at all four of these aspects for most systems. Previously, we discussed two aspects of the retina: anatomy and physiology. The Zachman (1987) framework uses the following aspects: Scope, Business Model, System Model, Technology Model, Detailed Representation and Real System (Bahill *et al.* 2006). These are aspects, not levels: a business model, for example, can have a lot of detail in it (Cohen 2003, Cohen and Wallace 2003). To help a team operate a complex system the documentation should address the following three aspects: equipment knowledge, task knowledge and team knowledge (Rouse *et al.* 1992).

We will now show an explicit example of many levels for two different aspects of a system. The following table shows two aspects of a softball game played by an NCAA Women Softball team.

Ten levels in two aspects of a softball game

Performance aspect	*Physical aspect*
One career	Expansion of universe
One season	Rotation of sun
One game	Rotation of moon
One inning	Rotation of Earth: Coriolis forces
One at-bat	Weather: Gulf Stream, barometric pressure
One pitch and subsequent activity	Atmosphere: lift, drag, temperature, humidity, winds
One pitch ⇐	⇒ Spin, speed and deflection
One collision	3D structure: four concentric shells
The sweet spot	Material: yarn, cork, rubber, horsehide
One vibrational mode	Atomic structure

These two aspects are orthogonal: contents of a box in the left column are not related to contents of a box in the right column, except at the level where they intersect: models for one pitch will be related to the spin, speed and deflection of the ball. A system should not contain models of the sweet spot and also models for the rotation of the moon. The levels in the decomposition of the performance aspect are independent of the levels in the decomposition of the physical aspect. Putting "One Career" and "Expansion of the universe" in the same model would make no sense. The only conjunction that makes sense is the intersection "One pitch" and "Spin, speed and deflection."

Naive students of physiology and anatomy are often confused, because they expect congruence between anatomical and physiological levels. But in general it does not exist. There is no one-to-one match in the levels that we have described for the retina of the eye.

Martin Glinz's (www.ifi.unizh.ch/~ glinz) four aspects of abstraction

Whole	*General*	*High-level*	*Type*
↑ composition	↑ generalization	↑ service	↑ classification
↓ decomposition	↓ specialization	↓ usage	↓ instantiation
Part	Special	Low-level	Instance

The UML uses the following diagrams to describe various aspects of a system: use case, use case diagram, communication diagram, sequence diagram, class diagram, state machine diagram, activity diagram and deployment diagram. The IDEF0 modeling scheme cleared up a lot of confusion when they separated inputs into two aspects: data and control.

Each aspect of a system will be decomposed into levels. But the levels for one aspect are independent of the levels of another aspect. A system has a physical decomposition, a functional decomposition and a requirements decomposition. It would be a mistake to allocate a high-level system function to a low-level part. It would be a mistake to link a detailed low-level requirement to a top-level system function. Developing requirements in the use cases helps keep the levels of the functions and the requirements the same (Daniels and Bahill 2004).

6. Generalizations about modeling

All components in a model should be at the same level, but models can be broken into submodels that are arranged hierarchically in levels. Models should only exchange inputs and outputs with other models of the same level, or maybe one level higher or lower. In Figure 6, models 0, 1 and 2 are at one level. They exchange information with each other. In this figure, Model 1 has its own functions, has interface definitions and it uses submodels 1.1, 1.2 and 1.3. These submodels exchange information with each other, with model 1 in the level above and with models 1.3.1, 1.3.2 and 1.3.3 in the level below. However, as shown in Figure 6, models should not skip levels in exchanging information. Feedback loops are not an exception to this rule.

Figure 6 applies to modeling, analysis, system design and organizational functioning. For all of these tasks it is important to pay attention to the levels of the communicating entities. For example, if Figure 6 were an organization chart, then person 1.3.1 would routinely communicate with 1.3.2, 1.3.3, 1.3.1.1, 1.3.1.2, 1.3.1.3 and 1.3. But if 1.3.1 were to communicate with 1.2, then he or she would have to be very careful to keep 1.3 informed, because the normal interfaces for that communication would not have been defined.

You cannot skip levels in the judicial system. If a policeman gives you a traffic ticket, you cannot appeal it immediately to the US Supreme Court. You must first go through all of the lower courts.

We now present a summary example of various levels in a task. This example is that of codebreaking and it comes from Kahn (1996).

Tasks in different levels for codebreaking

Level of abstraction	*Content*	*Comments*
Proof	Diplomatic tricks	Prove that your translation is correct
Translations	If Captain Dreyfus has not had relations with you …	Use the sentence and an Italian translator to get this translation
Sentences	Se Capitano Dreyfus non ha avuto relazione …	Use individual words to get this sentence
Words	*Dr*eyfus	Use the plaintext and the Baravelli Codebook to get this word
Plaintext	*227* 1 98 306	Use the ciphertext and the cipher to get this plaintext
Ciphertext (Telegram)	*527* 3 88 706	Use the international Morse code message and a transcription table to get this cipher text
International Morse code	●●●●● ●● – – – – – ●●● ●●● – – – – – ●● – – – ●● – – ●●● – – – – – – ●●●●	This is the original transmission

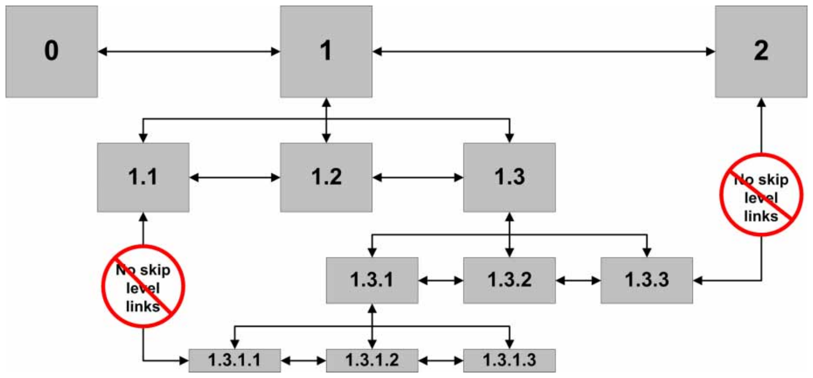

Figure 6. Models should be hierarchical. Models should not skip levels in exchanging information. Copyright ©, 2004, Bahill, from http://www/sie.arizona.edu/sysengr/slides/ used with permission.

Information goes up and down this hierarchy, but it cannot skip levels. The right column in the above table describes the upward process. An example going downward is that cryptographers found that *Dr* at the word level would be coded as *227* in the plaintext level. Then they discovered that *227* of the plaintext level corresponded to *527* of the ciphertext level. This helped them to discover the cipher that was used.

There are many aspects to codebreaking. In addition to the cryptology shown above, frequency analysis studies the rate of message traffic between network nodes. Spies aid the collecting of human intelligence. Photoreconnaissance missions gather imagery information. These aspects are all decomposed into their own independent levels.

Most systems are designed as hierarchies of components. Interconnecting components adds complexity, problems and concerns. Typically a model will be interested in its inputs, outputs, functions and states: for simplicity let's call these variables. When a model is interconnected with another model, it now becomes interested in *some* of the other model's variables: but it cannot get away with that. It must contend with *all* of the other models variables. This is what adds complexity. A second problem produced by interconnecting components concerns the way the variables are described. The variables of low-level models will typically be designed with high accuracy. Whereas variables in higher-level models will be described with statistical distributions characterized by the type, mean, variance and multiple modes, which are all more uncertain than the variables in the low-level models. Merging high-accuracy variables of the low-level models with uncertain summary statistics of the higher-level models is dangerous: it could destroy the validity of the low-level models. A third consideration needed because of interconnecting components concerns inputs and outputs. When low-level models are integrated into higher-level systems the inputs and outputs of interest are changed. The low-level models are concerned with the interaction of high-accuracy low-level inputs and outputs. Whereas the higher-level system (and the external world) ignores these low-level inputs and outputs and is only concerned with the higher-level inputs and outputs.

Resnick (1994) noted that many layered systems exhibit emergent behaviours. He documented how seemingly straightforward extrapolation from a lower level often fails to predict behaviour at a higher-level. This occurred in his simulation of an ant colony. No ant gives orders or tells other ants what to do. Instead, each ant reacts to stimuli and leaves behind a pheromone trail, which provides a stimulus to other ants. Each ant is an autonomous unit that reacts depending on its local environment and its genetically encoded rules. No single ant is intelligent, but ant colonies exhibit complex behaviour. Emergent behaviour means that the system is capable of doing something that was not planned in the design of the constituent parts.

It is a new concept in the system of systems architecture spawned by the DoD evolutionary acquisition life cycle model.

Wilber (2000) in his *Theory of Everything* organizes things into levels of holons. He says that if a level of holons is completely destroyed, then all of the levels of holons above it are also destroyed, but none of the levels of holons below are destroyed. For example, a lesion in the lateral geniculate nucleus will stop image processing in the visual cortex, but the cells in the retina will continue to function correctly. In codebreaking, if we eliminate the language translators, then we will not be able to understand the messages, but the cryptologists will still be able to continue their low-level work. In the CMMI, if the Integrated Project Management Process (level 3) were destroyed then Quantitative Project Management (level 4) would be impossible, but Project Planning (level 2) could continue.

We cannot describe in general what levels should be for systems that have evolved. However, in models where the levels were designed, we might be able to extract general principles. In this paper, we described two models where the levels were designed: the OSI seven-layer architecture model for telecommunication and the CMMI.

The CMMI (http://www.sei.cmu.edu/cmmi/) consists of industry best practices that address the development and maintenance of products and services over the total system life cycle (Chrissis *et al.* 2003). In the CMMI, the five levels of institutional maturity are defined. Then it is explained which of the 25 process areas should be in each level. A process area is a collection of related processes. Each process area is a different aspect of the company. Each process area has a description, specific goals and generic goals. The description contains a purpose statement, introductory notes and an explanation of related process areas. The specific goals are unique to each process area: they vary in number and complexity from process area to process area. The specific goals are achieved by performing specific practices, which are supported by typical work products and subpractices. The generic goals are grouped according to common features. The statement of each generic goal is the same for all process areas. The generic goals are achieved by performing generic practices that must be individually elaborated for each process area. The process areas are at different levels of abstraction. For example, Causal Analysis and Resolution (level 5) is a more abstract process area than Supplier Agreement Management (level 2).

A company implements a CMMI model by creating processes that satisfy each process area and assembling these into a model that describes how that company does business. This is an evolutionary procedure taking the company from level 1 through level 2, to level 3 and for a very well run company through level 4 all the way up to level 5. A particular process area would not be added to the company's model until after its lower-level process areas had already been included. A company's implementation of any individual process area is usually hierarchical.

Now, what general principles can we extract from the CMMI model? The levels are defined. Each process area is identified as belonging to a particular level. Each process area description states how each process is invoked. The model is arranged in a hierarchy and each process area is hierarchical. All process areas have the same general structure. However, process areas in the same level have greater similarities than process areas in different levels. The relationships between process areas are described. Of the 101 relationships that are listed, only 12 skip levels and these skip-level relationships are carefully defined. Processes at higher-levels are more abstract. The process areas are the aspects of the model.

Why is it hard to get the level right? (1) Getting the correct level for a model of a component is a complex activity, because, for example, each component could be in a different phase of its life cycle. Therefore, at any one time, different components could exist at different levels of detail. A model may need to contain abstractions of these components. So each component might need a different amount of abstraction. (2) There is no known general principle by means of which levels may be established. (3) Many aspects of each component must be modeled. Each aspect has levels and the levels of one aspect are different from the levels of another aspect. The model must be constructed using elements of the same level from the different aspects.

7. Summary

Most systems are impossible to study in their entirety, but they are made up of hierarchies of smaller subsystems that can be studied. Simon (1962) discussed the necessity for such hierarchies in complex systems. He showed that most complex systems are decomposable, enabling subsystems to be studied outside the entire hierarchy. For example, when modeling the movement of a pitched baseball, it is sufficient to apply Newtonian mechanics considering only gravity, the ball's velocity and the ball's spin (Bahill and Baldwin 2007). One need not be concerned about electron orbits or the motions of the sun and the moon. Forces that are important when studying objects of one order of magnitude seldom have an effect on objects of another order of magnitude.

Acknowledgements

We thank Dave Baldwin and Donna West for pointing out the Picasso abstraction sequences and Jerry Swain for the police officer example. This paper was supported by AFOSR/MURI F49620-03-1-0377.

Notes on contributors

Terry Bahill is a Professor of Systems Engineering at the University of Arizona in Tucson. He received his PhD in electrical engineering and computer science from the University of California, Berkeley, in 1975. Bahill has worked with BAE Systems in San Diego, Hughes Missile Systems in Tucson, Sandia Laboratories in Albuquerque, Lockheed Martin Tactical Defense Systems in Eagan MN, Boeing Information, Space and Defense Systems in Kent WA, Idaho National Engineering and Environmental Laboratory in Idaho Falls and Raytheon Missile Systems in Tucson. For these companies he presented seminars on Systems Engineering, worked on system development teams and helped them describe their Systems Engineering Process. He holds a US patent for the Bat Chooser, a system that computes the Ideal Bat Weight for individual baseball and softball batters. He received the Sandia National Laboratories Gold President's Quality Award. He is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), of Raytheon and of the International Council on Systems Engineering (INCOSE). He is the Founding Chair Emeritus of the INCOSE Fellows Selection Committee. His picture is in the Baseball Hall of Fame's exhibition "Baseball as America." You can view this picture at http://www.sie.arizona.edu/sysengr/

Ferenc Szidarovszky is Professor of Systems & Industrial Engineering at the University of Arizona in Tucson. He was born in Budapest, Hungary and received his B.S., M.S. and PhD in numerical techniques from the Eötvös University of Sciences in Budapest. He received a second PhD in economics from the Budapest University of Economic Sciences in 1977. He was an assistant and an associate professor in the Department of Numerical Analysis and Computer Sciences of the Eötvös University of Sciences. He served as the Acting Head of the Department of Mathematics and Computer Sciences of the University of Horticulture and Food Industry. He was a professor with the Institute of Mathematics and Computer Sciences of the Budapest University of Economic Sciences.

Rick Botta is the Director of Systems Engineering for BAE Systems in San Diego. He holds a Bachelor of Science degree in Computer Science from California Polytechnic State University, San Luis Obispo. Rick has a quarter century experience in a wide variety of engineering, engineering management and program management roles involving development and integration of complex, software intensive systems. He is a member of INCOSE.

Eric Smith earned a B.S. in Physics in 1994, an M.S. in Systems Engineering in 2003 and his PhD in Systems and Industrial Engineering in 2006 from the University of Arizona, Tucson. He is currently a Visiting Assistant Professor in the Department of Engineering Management and Systems Engineering, at the University of Missouri at Rolla, 65409.

References

Abadi, C.D. and Bahill, A.T., 2003. The difficulty in distinguishing product from process. *Systems Engineering*, 6 (2), 108–115.

Bahill, A.T. and Baldwin, D.G., 2007. Describing baseball pitch movement with right-hand rules. *Computers in Biology and Medicine*, 37, 101–108.

Bahill, A.T. and Dean, F.F., 2007. Discovering system requirements, Chapter 4. In: A. Sage and B. Rouse, eds. *Handbook systems engineering and management*. 2nd ed. New York: John Wiley & Sons.

Bahill, A.T. and Henderson, S.J., 2005. Requirements development, verification and validation exhibited in famous failures. *Systems Engineering*, 8 (1), 1–14.

Bahill, A.T., Botta, R. and Daniels, J., 2006. The Zachman framework populated with baseball models. *Journal of Enterprise Architecture*, 2 (4), 50–68.

Botta, R., Bahill, Z. and Bahill, A.T., 2006. When are observable states necessary? *Systems Engineering*, 9 (3), 228–240.

Boeke, K., 1957. *Cosmic View, the Universe in 40 Jumps*. New York: John Day Company.

CISCO, 2006. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/osi_prot.htm, accessed October 2006.

Chrissis, M.B., Konrad, M. and Shrum, S., 2003. *CMMI: Guidelines for Process Integration and Product Improvement*. Boston: Pearson Education Inc.

Cockburn, A., 2001. *Writing Effective Use Cases*. Reading: Addison-Wesley.

Cohen, R.B., 2003. Teaching note on A-Rod: signing the best player in baseball. *Harvard Business School Case Study*, 5-203-091.

Cohen, R.B. and Wallace, J., 2003. A-Rod: signing the best player in baseball. *Harvard Business School Case Study*, 9-203-047.

Comer, D.E., 2004. *Computer Networks and Internets with Internet Applications*. Pearson: Prentice Hall.

Daniels, J. and Bahill, A.T., 2004. Hybrid process combines traditional requirements and use cases. *Systems Engineering*, 7 (4), 303–319.

Eames, C. and Eames, R., 1968. *A Rough Sketch for a Proposed Film Dealing with the Powers of Ten and the Relative Size of the Universe*, Presented at the annual meeting of the Commission on College Physics, 8 minutes, color.

Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Reddy, D.R., 1980. The Hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12 (2), 213–253.

Evans, E., 2004. Domain-driven design: Tackling complexity in the heart of software. Boston: Addison-Wesley.

Ferber, J., 1999. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. ISBN 0-201-36048-9. Harlow: Addison Wesley Longman.

Feynman, R.P., 1985, Surely you are joking, Mr Feynman. New York: WW Norton.

Hooks, I.F. and Farry, K.A., 2001. *Customer-Centered Products: Creating Successful Products through Smart Requirements Management*. Chapter 9. New York: AMACOM.

Kahn, D., 1996. *The Codebreakers: The Story of Secret Writing*. NY: Scribner.

Kang, T. and Mavris, D.N., 2005. A systems-of-systems approach for application to large-scale transportation problems. *Proceedings of 15th Annual International Symposium of INCOSE*, Rochester, NY, July 10–14.

Kaufman, P.L. and Alm, A., 2002. *Adler's Physiology of the Eye*. 10th ed. St Louis: Mosby.

Kerzner, H., 2003. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. 8th ed. New York: Van Nostrand Reinhold.

McCloud, S., 1993. *Understanding Comics: The Invisible Art*. Northampton, MA: Tundra. Lett. Bob Lappan., page 31, ISBN 1-56862-019-5. Available online at: http://www.scottmccloud.com/store/books/uc.html.

Miller, G.A., 1956. The magical number seven, plus or minus two: some limits on our capacity for processing information. *The Psychological Review*, 63, 81–97. Available online at: www.well.com/user/smalin/miller.html.

Morganwalp, J. and Sage, A., 2002/2003. A system of systems focused enterprise architecture framework and an associated architecture development process. *Information, Knowledge, Systems Management*. Amsterdam: IOS Press, Vol. 3, 87–105.

Morrison, P. and Morrison, P., 1977. The Powers of Ten. In: W.H. Freeman, ed. *Scientific American books*, This book has been converted into many web sites including http://www.wordwizz.com/pwrsof10.htm.

National Security Agency, 2006. http://www.nsa.gov/sigint/ [Accessed August 2006].

OMG, 2006. http://www.uml.org/ [Accessed August 2006].

Övergaard, G. and Palmkvist, K., 2005. *Use Cases: Patterns and Blueprints*. Indianapolis: Addison-Wesley.

Picasso, P., 2006. http://mourlot.free.fr/fmtaureauenglish.html [Accessed August 2006].

Resnick, M., 1994. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge, MA: The MIT Press.

Rouse, W.R., Cannon-Bowers, J.A. and Salas, E., 1992. The role of mental models in team performance in complex systems. *IEEE Transactions on Systems, Man and Cybernetics*, 22 (6), 1296–1308.

Simon, H.A., 1962. The architecture of complexity. *Proceedings of the American Philosophical Society*, 106, 467–482.

Szidarovszky, F. and Bahill, A.T., 1998. *Linear Systems Theory*. Boca Raton, FL: CRC Press Inc.

Tufte, E.R., 1997, Visual explanations: Images and quantities, evidence and narrative. Cheshire: Graphics Press.

Waltz, E. and Llinas, J., 2000. *Multisensor Data Fusion*. Boston: Artech House.

Warwick, R., 1976. *Eugene Wolff's Anatomy of the Eye and Orbit*. 7th ed. Philadelphia: Saunders.

Wilber, K., 2000. *A Theory of Everything: an Integral Vision for Business, Politics, Science, and Spirituality*. Boston: Shambhala Publications.

Wymore, A.W., 1993. *Model-Based Systems Engineering*. Boca Raton, FL: CRC Press.

Zachman, J.A., 1987. A framework for information systems architecture. *IBM Systems Journal*, 26 (3). Available online at: http://researchweb.watson.ibm.com/journal/sj/382/zachman.pdf [Accessed 30 April 2007].

Zeigler, B.P., Praehofer, H. and Kim, T.G., 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. San Diego: Academic Press.