Introduction to Structured Methods

The idea that system descriptions are more clear and easier with pictures rather than words provided the basis for the development of structured methods. Structured analysis techniques, developed in the 1970's by Edward Yourdon [1], Tom DeMarco [2] and others, improved the system specification process. Derek Hatley and Imtiaz Pirbhai [3] have built on this work to develop a comprehensive method for specifying real-time systems.

The Hatley-Pirbhai methodology (HPM) is a graphical method used to specify both a system's function and its physical partitioning. To this end, HPM separates the system specification into two models: requirements and architecture. The requirements model describes what the system should do (its function), while the architecture model describes the physical entities in the system and the allocation of requirements to these entities. The process of allocating functions to physical entities provides an excellent mechanism for ensuring interface consistency. Significant improvements in interface consistency are one of the more commonly cited benefits of HPM. Table 1 lists other features and benefits of the method.

Table 1 HPM Features and Benefits. The rigor and hierarchical nature of HPM provide specific benefits.

Features	Benefits
Hierarchical model	• Specifies requirements at appropriate level • Depicts manageable amount of information at one time
Graphical and text representation of functionality	• Clearly shows interfaces (functional and physical) • Graphics depict abstract aspects of system • Text defines details
Allocation of functions to physical entities	• Greatly improved interface consistency
Rigorous method	• Promotes thorough design • Identifies gaps early

The value of separating requirements from architecture is that it forces systems engineers to identify the system's essential functions without unnecessarily constraining the implementation. This encourages the systems engineer to leave the design detail decisions to the designers who know best how to use available technology to achieve this system's requirements.

The process provides structure and rigor to the activities normally performed by engineers. Developing requirements hierarchically from system, to subsystem, to module, to component structures the requirements in such a way that permits engineers to specify requirements at the appropriate level. It encourages rigorous top-level decisions in the beginning and pushes detailed decisions to later stages in the project as appropriate. The model hierarchy also organizes a complex set of requirements and depicts a manageable amount of information at one time.

The combination of graphics and text to represent the system functionality exploits the strength of pictures to depict abstract aspects of the system with the support of text to define the details. Critical for the development of complex systems, the graphical approach also clearly shows the functional and physical interfaces. Early focus on interfaces promotes a thorough understanding of the system boundary and a thorough design, minimizing chances of problems later in development.

Implementing the system development process with HPM reduces the number and magnitude of problems during integration and test of a new system. This occurs because the rigor and consistency checking provide virtual integration and test, identifying design problems before building the system. Structured methods provide rigor in the front-end to eliminate problems when they are inexpensive to fix.

PO Box 2875 • El Segundo, CA 90245 • Phone: (310) 607-9609 • Fax: (310) 607-9475 • consult@hasys.com • http://www.hasys.com/

Requirements Model

The requirements model combines traditional structured analysis and finite state modeling techniques to define the system function. A hierarchical set of data flow diagrams (DFDs) and process specs provides the basis of the system specification (see Figure 1). A DFD consists of a set of labeled circles (called processes), pairs of parallel lines (called stores), and labeled arrows (called data flows). The data flows enter and exit the process and stores. Each process transforms the input data flows into the output data flows. A store is data frozen in time, making it continuously available whether the input is available or not.

The top-level DFD, called the context diagram, describes the system's functional boundaries and functional interfaces. It contains one process and the external entities with which the system interacts, shown as rectangles (called terminators) as shown in Figure 1. The context diagram is a valuable tool for communicating what is in the system and what is not. The next level down DFD (called DFD 0) models the process on the context diagram (see Figure 1). Lower level DFDs provide more detailed descriptions of each process unless it is a primitive process. A primitive process is one simple enough that further decomposition provides no value. Process specs provide short, approximately one-page, descriptions of primitive processes.

To specify the system's finite state machine behavior, HPM adds control flow diagrams (CFDs) and control specs to the DFDs and process specs (see Figure 1). There is a CFD for every DFD, however, most computer-aided software engineering (CASE) tools combine the DFD and CFD into one diagram and provide options to view data, control, or both. Each CFD contains exactly the same processes and stores as its corresponding DFD. However, the CFD shows control flows instead of data flows and adds control spec bars. Control flows entering or exiting a control spec bar are the inputs or outputs of the corresponding control spec. Control specs use traditional finite state machine descriptions such as decision tables and state transition diagrams. The control spec describes the transformation of input control flows into output control flows and the logic for enabling or disabling the processes on the CFD (and its corresponding DFD).

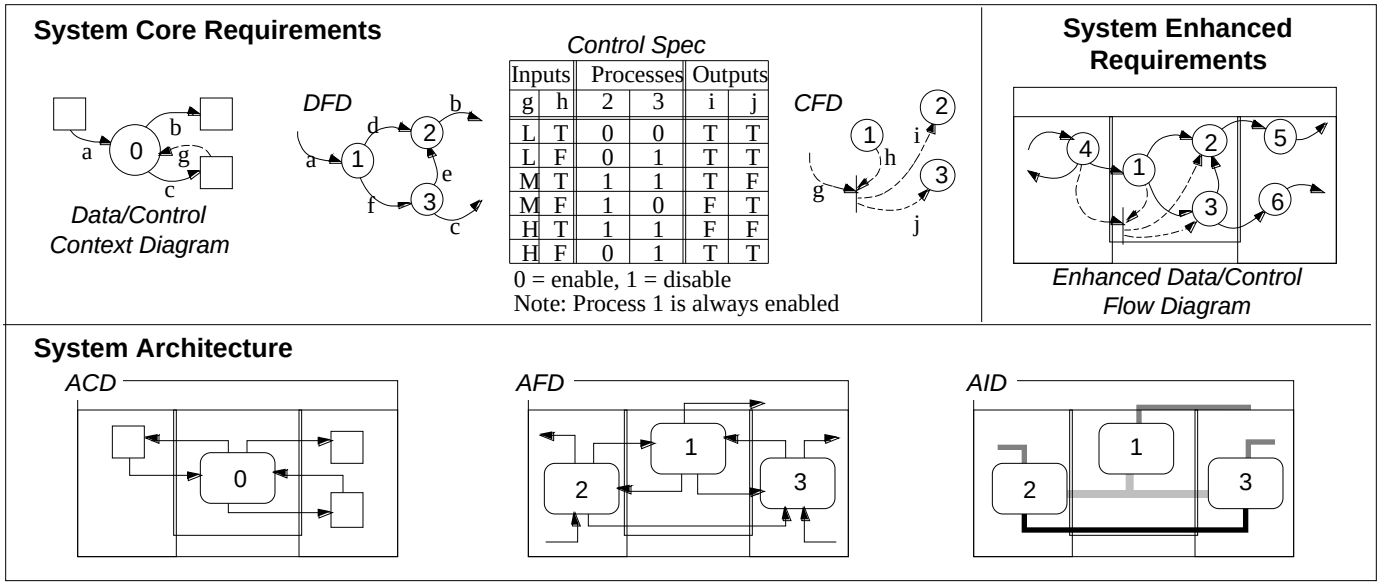

Figure 1 Model Diagram Examples. The model diagrams depict the system functionality in the core requirements model and the physical partitioning of the system in the architecture model. The enhanced DFD captures technology-dependent functionality.

PO Box 2875 • El Segundo, CA 90245 • Phone: (310) 607-9609 • Fax: (310) 607-9475 • consult@hasys.com • http://www.hasys.com/

An important aspect of HPM is consistency checking. All flows entering a process or control spec bar on a diagram must appear in the DFD, CFD, process spec or control spec for the process and vice versa. Also, all flows in the model must appear in the dictionary, including the definitions for elements in the flow decompositions. An automated tool can perform consistency checking and identify discrepancies and omissions in the model. The process of generating and defining consistent models forces the system designer to justify the information flowing into and out of the system as well as between processes. This identifies both missing and unnecessary flows, eliminating errors early in the system development process.

The technology-dependent aspects of the system are more likely to change during system development as technology advances. Isolating technology-dependent aspects from the core function of the system minimizes changes to the interface descriptions and requirements due to exploitation of new technology. HPM achieves this isolation with the core and enhanced requirements models (see Figure 1). The core model captures the core requirements of the system, which are independent of specific technology. The enhanced requirements model adds around the core model requirements for input, output, user interface and maintenance/self test processing. An architecture template provides separate regions for each of the four typical categories of enhanced requirements.

Architecture Model

The architecture model, derived from functional block diagrams, describes the physical partitioning of the system (see Figure 1). The architecture context diagram (ACD) shows the physical boundaries of the system. The architecture flow diagram (AFD), shows the physical entities, called modules, in the system. An architecture module may be a system, subsystem, unit, unit module, component, etc., depending on the level of detail of the model. The architecture flows on the AFD show the transfer of information between modules. The partitioning of the DFDs and CFDs to architecture modules determines the composition of the architecture flows. The architecture dictionary specifies the combinations of data and control flows that define each architecture flow.

The architecture interconnect diagram (AID) depicts the interconnection of modules in the system (see Figure 1). The architecture interconnects represent the physical channels, such as electrical wires, busses, waveguide, optical cable, or mechanical connections, through which modules exchange information. The type of line used to represent an interconnect depicts the type of channel. The AFDs and AIDs are pairs of diagrams each showing the same modules. The architecture dictionary captures the allocation of architecture flows to interconnect channels and provides traceability for interface definition.

Each module in the AFD has a subset of the enhanced requirements model allocated to it. Superbubbles placed around groups of processes, control specs and stores on the enhanced data/control flow diagram (EDFD) indicate the module designated to perform those functions. Further decomposition of processes partially allocated to a module satisfies the requirement that whole processes be allocated to modules. The traceability matrix documents the allocation of requirements to architecture.

Model Development Process

The development process of the layers of the models, illustrated in Figure 2, is as follows:

1. Develop system core requirements including context diagram, DFDs, CFDs, control specs, process specs and dictionary (Shown are context diagram and top-level DFD).
2. Enhance the core requirements.

PO Box 2875 • El Segundo, CA 90245 • Phone: (310) 607-9609 • Fax: (310) 607-9475 • consult@hasys.com • http://www.hasys.com/

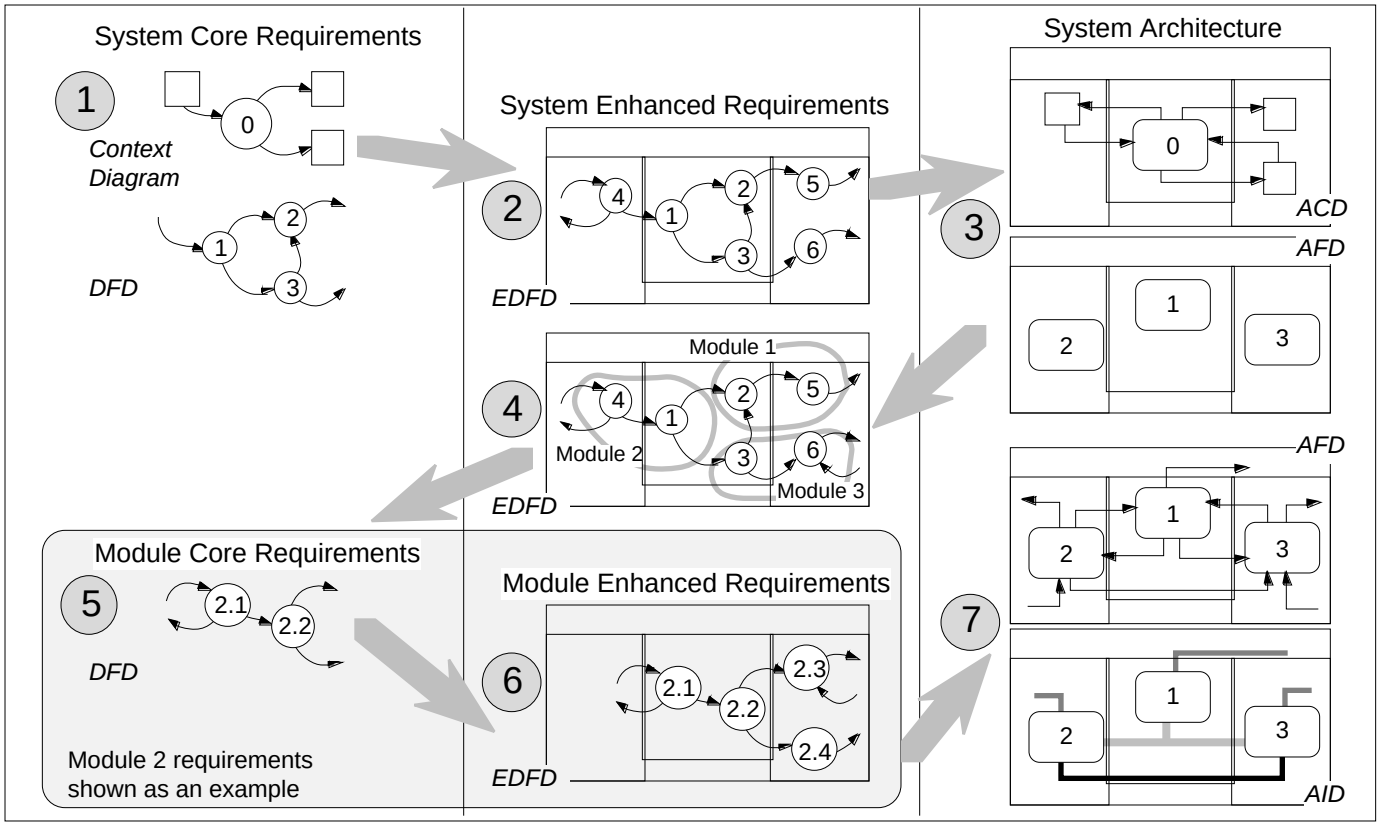

Figure 2 Model Development Process. Enhancing requirements models for each architecture module before drawing flows on the AFD provides consistent allocation of architecture flows to interconnects.

3. Determine the architecture modules for the system and draw the ACD (Shown are the ACD and the AFD without flows).
4. Allocate processes, control specs and stores from the enhanced requirements to architecture modules using superbubbles and the traceability matrix.
5. Draw the core requirements for each module in the system, taking the requirements directly from the traceability matrix (Shown are module 2 core requirements with processes 4 and 1 from the system renumbered as 2.1 and 2.2, respectively).
6. Enhance the module core requirements as necessary for intermodule communication, new user interfaces and maintenance/self-test (Note: Because 2.1 is an enhanced process from the system model, there are no enhanced processes for the inputs).
7. Draw the flows and interconnects on the AFD and AID based on the boundaries of the enhanced module requirements (Shown are the AFD and AID).

This process, developed from experience on complex systems [4], resolves some inconsistencies in the original methodology. Specification of a complex system proceeds one layer at a time with more detail being added at each level. Ultimately, the allocation of the requirements model to the architecture model results in the partitioning of each model component to either hardware or software.

PO Box 2875 • El Segundo, CA 90245 • Phone: (310) 607-9609 • Fax: (310) 607-9475 • consult@hasys.com • http://www.hasys.com/

References

[1] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*., Prentice-Hall, Englewood Cliffs, 1979.

[2] T. De Marco, *Structured Analysis and System Specification*., Prentice-Hall, Englewood Cliffs, 1978.

[3] D. J. Hatley and I. A. Pirbhai, *Strategies for Real-Time System Specification*., Dorset House, New York, 1988.

[4] J. A. Rader and L. J. Haggerty, "Supporting Systems Engineering with Methods and Tools: A Case Study," presented at the *28th Asilomar Conf. Signals, Systems, Computers*, Asilomar Conf. Ctr., Pacific Grove, CA, Oct. 31-Nov. 2, 1994.

Biographies

Kip Haggerty, Ph.D., P.E. has over 20 years of experience in systems engineering. He received the Ph.D. degree in electrical engineering from the University of California, Los Angeles in 1988.

Since August of 1994, he has been a consulting engineer with H&A Systems Engineering and has used HPM to specify requirements for a multiple-unit parallel solid-state power conversion system. He developed and taught a hands-on Kalman Filtering course for the Naval Weapons Center.

From 1980 to 1994, he was a member of the technical staff, systems engineer, and then senior staff engineer at Hughes Aircraft Company, El Segundo, CA. While at Hughes, he led system concept studies on several programs and used HPM on two programs to define system requirements and develop system architectures. His systems engineering experience covers a broad range of sensor and communication technologies as well as control and signal processing algorithms. He developed and taught a Kalman Filtering course for Hughes.

Dr. Haggerty is a senior member of IEEE and a member of Tau Beta Pi. He is a licensed electrical and control systems engineer in the state of California and has five published papers.

Leslie Haggerty, B.S. has over 14 years of experience in systems engineering. She received the B.S. degree in engineering from the University of Redlands, Redlands, CA in 1986.

Since August of 1994, she has been a consulting engineer with H&A Systems Engineering and has used HPM to specify requirements for a multiple-unit parallel solid-state power conversion system. She has taught two courses of the structured analysis portion of HPM.

From 1986 to 1994, she was a member of the technical staff and systems engineer at Hughes Aircraft Company, El Segundo, CA. While at Hughes, she led CASE tool implementation on one program and used HPM on two programs to define system requirements and develop system architectures. She taught the "Structured Systems Development," course developed by Imtiaz Pirbhai, for the Hughes technical education program and improved the course materials. She also had two approved invention disclosures.

Ms. Haggerty is a senior member of IEEE and a member of Phi Beta Kappa. She is a licensed electrical engineer in the state of California has published a joint case study paper on supporting systems engineering with CASE methods and tools.

PO Box 2875 • El Segundo, CA 90245 • Phone: (310) 607-9609 • Fax: (310) 607-9475 • consult@hasys.com • http://www.hasys.com/