Edgar H. Sibley
Panel Editor

A whole-system designer, fire fighter, mediator, and jack-of-all-trades, the system architect brings unity and continuity to a development project—offsetting the inevitable compartmentalization of modern modular designs.

A PRAGMATIC VIEW OF THE SYSTEM ARCHITECT

JOHN A. MILLS

This paper provides a pragmatic view of the role of the system architect based on the author's own experience as a system architect on three separate system projects during the past six years. One system was a batch forecasting system that manipulated large data and reference files and processed sophisticated routing and forecasting algorithms. The second, an on-line, application-level communication system, interfaced between a very large, on-line system and other application systems or other copies of the system itself, providing routing, message translation, error correction and recovery, and message deferral. The third—perhaps one of the largest commercial systems in existence today—is the large, on-line system referred to above, the Trunk Integrated Records Keeping System (TIRKS™),[1] a set of integrated on-line and batch systems providing a wide range of operations support functions.

In each case, design work on the system had begun before the author was assigned to it as system architect, and there was already a commitment, to some extent, to certain designs, features, and interfaces. Thus the experience upon which this paper is based is that of the system architect responsible for directing and unifying a *developing* system. No attempt is made in this paper to be theoretical. Rather, it is hoped that the experience presented here has not been unusual and will prove useful to the designer newly appointed as system architect and to managers deciding whether such a position is required for their system and how it should be organized.

THE SYSTEM ARCHITECT AND THE DEVELOPMENT ORGANIZATION

Reaching Critical Mass—Why a System Architect?

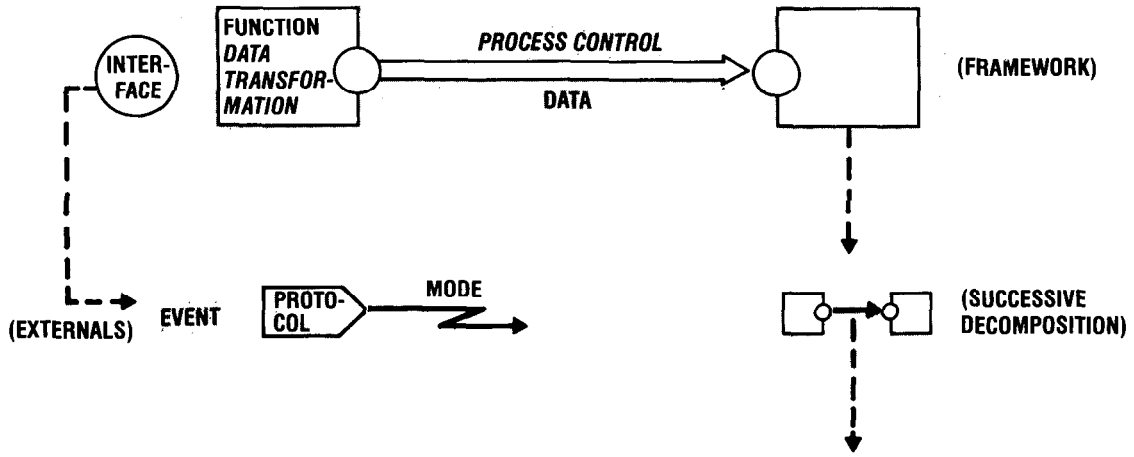
A system design consists of four parts: *function*, *data*, *process control*, and *data transformation*; plus an interface that consists of the *event* causing data to be passed through the interface, the *protocol* in which the data are encoded, and the *mode* of communication (e.g., dial-up, tape) (see Figure 1). A design in progress is successively decomposed into modules; each module's design can be described by the four parts of the design framework and the three parts of an interface. The highest level of design is the *framework* (the four parts) and the *externals* (the interfaces) of the successive layers of modules. The architecture design is the design of this highest level.

A simplified view of the steps through which a designer progresses to implement a system is given in Figure 2. Each step is a successive level of refinement directed toward the writing of code: A user makes a request, a system engineer writes requirements, a system architect develops an architecture design, a system designer develops a system design, a designer develops a detailed design, and a coder codes. The process is like a funnel. The initial steps require a broad knowledge and a broad view, gradually narrowing to a concerted effort to produce code. Sliding out at the end is a running system.

These steps can be applied to an organization recursively in terms of the different organizational levels affected. For example, an organization developing large application systems might have a way of receiving user requests from outside; then, it might have a requirements division and an architecture division, with system-design and detailed-design efforts spread over a number of development groups. However, all these steps might also apply to the individual development group.

The problem is deciding how this process should best be organized. The software-design process is an integrated procedure, where each step runs into the next, each feeding back to previous steps. The difficulty is defining the precise responsibilities at the higher level versus the responsibilities at the lower steps (*turf*). How does the person working the higher step communicate with those working below, and vice versa (*feedback*)? At

[1] TIRKS is a trademark of Bell Communications Research, Inc.



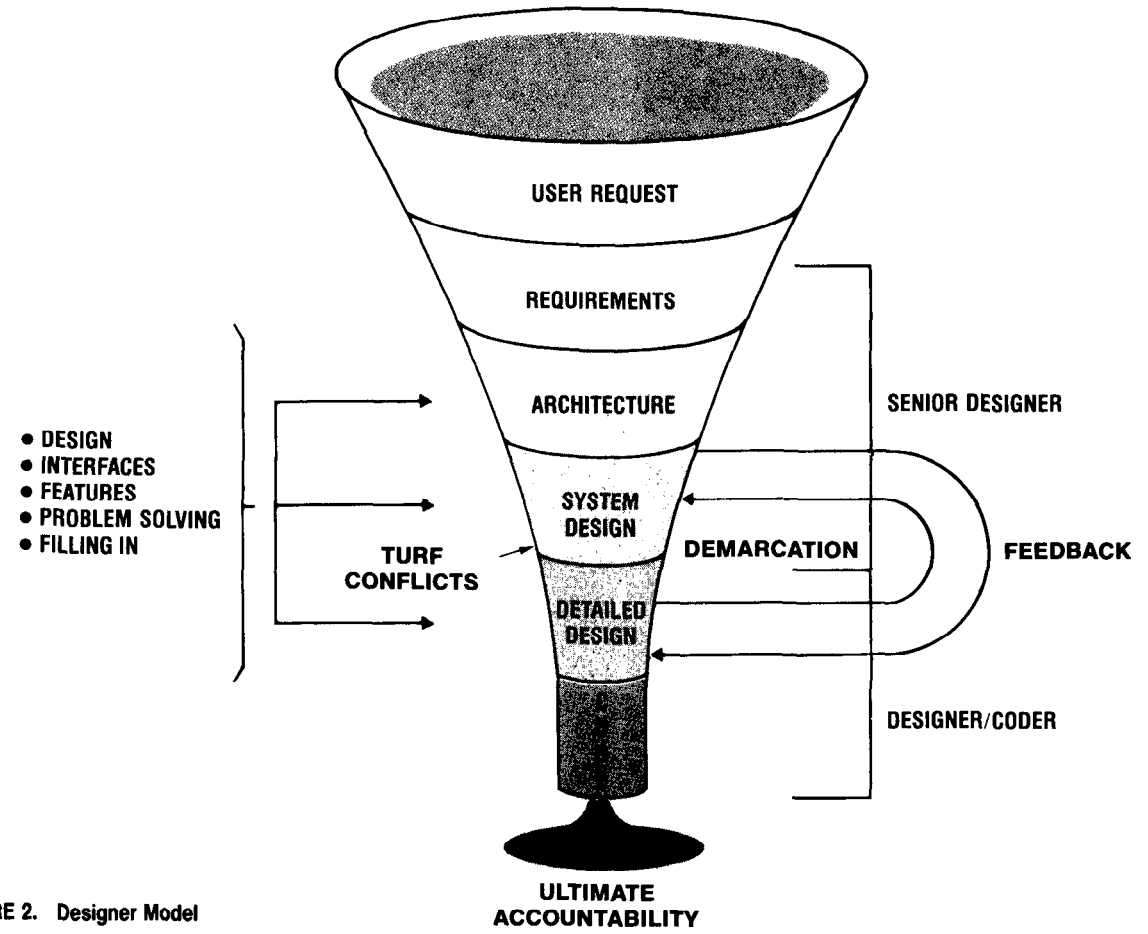


FIGURE 2. Designer Model

what point is the higher step finished (*handed off*)? How is the person responsible at the higher level held accountable for the work done there (*accountability*)?

Since accountability is measured best when the system is running (or failed), this latter issue becomes more difficult the higher one is in the development process.

Typically a division of labor occurs between the system-design step and the detailed-design step. The system designer hands a design off to a detail designer signaling the active end of the system-design phase. When the system designer and detail designer are in the same implementation group, which is normally the case, each is easily held accountable for the success of the system. However, the system-architecture and the system-design steps are traditionally done by the same person—a senior designer who also manages the development team as chief programmer.[2] In this case, the point at which the architecture step ends and the system-design step begins is not an issue. However, as the size and complexity of a system increase, a critical point is reached where system-architect responsibilities and system-design responsibilities cannot be held by the same person.

A large-system development effort will ordinarily have many senior designers (Figure 3). Developing an architecture and subsequently resolving architectural problems by a committee of all these designers would be both inefficient and unwieldy. The critical mass is typically reached when there are four or more interacting system designers.[3] Where this mass is exceeded and one senior designer does not emerge as the system architect, system-architect responsibilities are often sacrificed as they are less concrete and harder to grasp than the solidity of a running, functional (albeit possibly defective) system.

System Architect versus System Designer

The system architect and system designers establish a variety of working relationships: the *lead* relationship, where the system architect has primary responsibility for a design in consultation with the system designers; the *team* relationship, where the system architect and the system designers develop a design together; and the *consulting* relationship, where the system designers develop the design in consultation with the system architect.

This paper advocates the use of the first and last relationships, in combination. That is, the system architect should have responsibility for the design at a certain stage of development (the lead relationship); once this stage is completed, system designers should assume responsibility for subsequent stages (in a consulting relationship with the system architect). Thus the organizational issues of turf, feedback, handoff, and accountability are answered in the context of these two ordered relationships.

Turf. The system architect designs the framework and externals of a system [1, 2] and guarantees its integrity (i.e., the lead relationship); the framework is the high-level definition of the four parts of system design, and the externals the high-level definition of the three parts of the interface. A development group is generally authorized to develop a design starting at some stage of definition. A high-level definition (the top of this stage) supplies a definition of *what* the system should do for the next level down. Guaranteeing system integrity requires *surveillance* and *filling in* wherever something needed is not being done; it is often enforced by such means as architecture-design reviews.

The system designer picks up where the system architect leaves off. He or she designs the internals of the system in what is now a consulting relationship with the system architect. Typically, each system designer is responsible for one component defined in the framework.

Feedback. As the system architect develops a design for the framework and externals or a solution to an architectural problem, system designers are involved as consultants and reviewers (the lead relationship). The system architect publishes the results in a set of documents "delivered" to system designers. Once the deliverables are handed off to the designers, the roles reverse and the architect reviews the designs of the designers. The system designers do the developing, and the architect is a consultant and reviewer (the consulting relationship).

Handoff. A system architect hands off architectural deliverables to one or more system designers who reach an agreement with the system architect—via the various feedback loops—on when they have sufficient input to start their job. The boundary between the system-architect and system-design steps will vary according to the individual expertises involved (and thus when the relationship changes will vary as well). Therefore, although the system architect's turf is defined, flexibility is needed in marking its end.

Accountability. A system architect often does not remain with a system through its implementation. The period of time in the development life cycle during which the system architect's presence is essential is the analysis phase through most or all of the design phase (Figure 4). By the end of the design phase, most issues of concern to a system architect are resolved; system designers are well versed in the system's direction and philosophy so the architect's function drops off to a low level of activity (i.e., the consulting relationship is no longer necessary). By the time the system is running (or not running), the architect's impact on the system may be long forgotten. How then is the system architect held accountable for what he or she has done? Management should be able to evaluate the system architect's contribution by auditing his or her input into the final running system.

Symbiosis. The system architect's design functions enlist a *broad*, global, whole-system view, whereas the

[2] A chief programmer in a chief programming team manages the team, designs and codes, integrates top-level and critical modules, and reviews code [3].

[3] In an organization *without* a system architect, if x is the number of senior designers, and y the number of lines of communication, then $y = x(x - 1)$. In an organization *with* a system architect, if x is the number of system designers, and z the number of lines of communication, then $z = 2x$. Now,

$$y > z$$
$$x(x - 1) > 2x$$
$$x^2 - x > 2x$$
$$x^2 > 3x$$
$$x > 3.$$

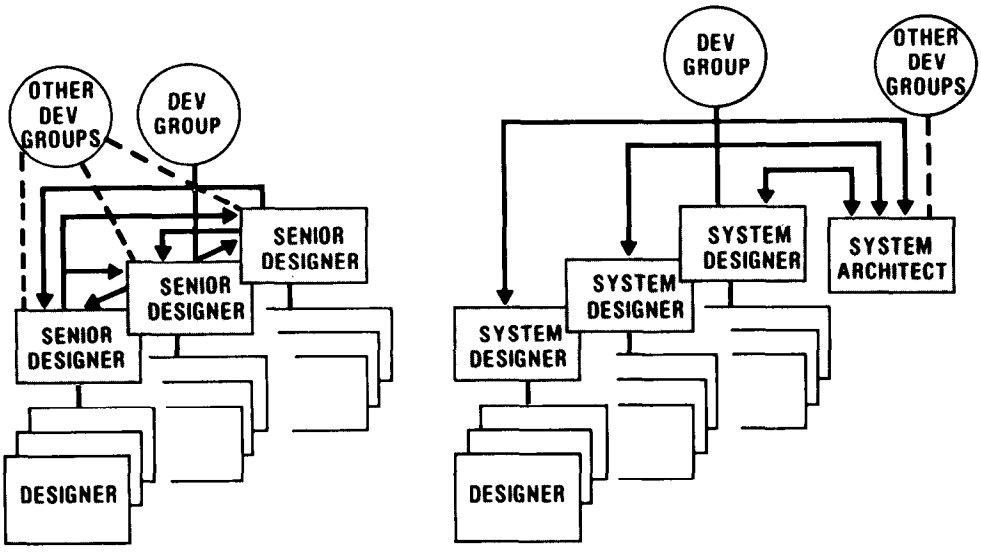


FIGURE 3. Critical Mass

system designer works from a component view and from a bottom-up perspective. Of course, the two views overlap and interact, the architect view being one of *breadth*, and the designer view one of *depth*. Both system architect and system designer *design*, but the architect takes a breadth approach emphasizing a bias toward the whole system, whereas the system designer emphasizes a component bias. With a grasp of all parts of a system, the system architect becomes a mediator and a negotiator; he or she designs and resolves issues in a negotiated way with all impacted system designers.

Deliverables. The deliverables from a system architect are documents: architecture-design specifications, next steps, standards, meeting minutes from the periodic reviews, and design audits of the running system.

Three Organizations with System Architects

The three systems alluded to in the introduction were each a different size. The on-line communication system, a *small* system developed by a group of five system designers, is a component in the *large* on-line system that had over 15 development groups. The batch system was *medium* sized with 2 development groups, 1 human-factors group, 1 system-engineering group, and 1 system-test group, and was unrelated to the other two.

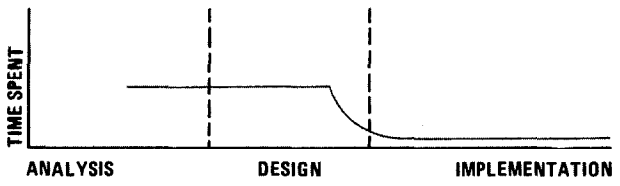


FIGURE 4. The System Architect in the Development Cycle

All three sample organizations are mixed organizations [1] in the sense that certain groups are responsible for distinct development functions and other groups are responsible for distinct projects. For example, system engineering and system test are functions performed by distinct groups. In the medium-sized system, the human-factors function is also a distinct group. In the large system, the system-architecture function is also a distinct group. However, in all three cases, the development groups are project oriented: They perform all the design and implementation functions for a given component and are ultimately responsible for the project's success.

Small-System Development. In the small-system development hierarchy (Figure 5), which is headed by a supervisor, system designers coordinate the work of designers; the system architect, like the system designers, reports to the supervisor. The system architect works with system designers to develop the system, and both influences and is influenced by the system designers. The system architect is the person consulted by the supervisor on systemwide problems; he or she also interacts with other development and user groups to define interfaces and functional capabilities and resolve problems. Typically, the system architect does a lot of organizing, studying, discussing, and mediating, but little actual implementing.

In the small-system case, the system architect establishes the overall framework of the system and refines the interfaces with other components in the large system and with external systems. The feedback and handoff is informal; agreement is reached on a personal basis, which is successful since the group is relatively small. Similarly, accountability is clear since the architect is mandated to ensure the system is successful.

Medium-Sized System Development. The development groups and the human-factors group in the medium-sized system were organized in the same way as the

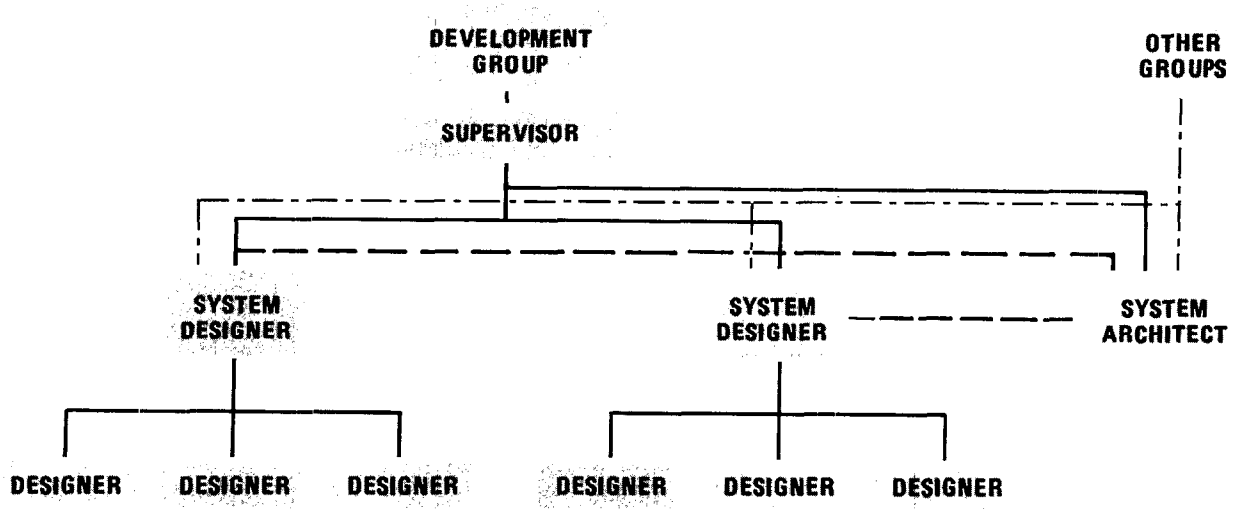


FIGURE 5. The System Architect in the Small–Medium-Sized Development Group

small-system group except that the system architect in one of the development groups was designated system architect for the entire system. Among system-architect responsibilities were tactical problems spanning all the groups, a variety of strategic topics (e.g., new-technology implementation), the system-architecture document, a design audit, and architectural issues in the architect's home group. The fact that each group had a system architect *and* one of these was made "more equal" than the others tended to blur the distinction between group system architects and the overall system architect; this was corrected by instituting a design-review committee made up of the group supervisors and the overall system architect. This committee received design problems and assigned them to appropriate people for resolution. By this procedure, turf and accountability were defined, and handoff decided. The committee also did a limited amount of design review.

Large-System Development. The large-system development effort consisted of many interrelated components developed by the same organization in the same development environment, and generally on the same operating system. In this case, an entire group was designated to perform the functions of system architect (Figure 6). Although the topologies for Figures 5 and 6 are virtually identical, the large-system organization also designates a *feature coordinator* for each new system feature. The feature coordinator is a supervisor whose development group has a major portion of responsibility for feature development and who is responsible for a feature's implementation from start to finish, including portions for which he or she does not have immediate, supervisory responsibility.

The system-architect group, since it is dealing with the integrity and future direction of the system as a whole (a very large universe), is unable to perform the whole range of system-architect responsibilities for each feature, which in turn becomes the responsibility of the feature coordinator. Instead, the system-architect group takes on special topics that span many groups, and are architecture related and of short duration. The system-architect group gets events started and then hands off to a feature coordinator; once handed off, architecture-design reviews are held to review the feature design. Architecture-design reviews are also held for features with which the system-architect group did not initially get involved. The system-architecture group sets up formal meetings to review architecture designs, the output of which is a list of open issues that have been assigned to respective supervisors. The system-architecture group then tracks the resolution of open issues, usually by holding follow-up meetings to review resolutions. Open issues of an architectural nature are assigned to the system-architect group.

Note the recursive nature of this organization: Within each development group, there is a system architect among the designers as there is a system-architect group among the development groups. In this larger organization, there is no clear measure of accountability for the system-architect group, since the proof of success, a working system, is very distant in time and form from the system-architect deliverables.

Recommendations

For a development project that needs a system architect, issues of turf, handoff, feedback, and accountability must be resolved in terms of the size and complexity of the system in question. In a small-enough system, the system architect should be able to deal with all architectural problems that arise. Handoff and feedback can be done informally, and accountability is clear, since the architect's input has direct and immediate impact on the final product.

In a system that requires several development and functional groups, more formality is required. There should be a system architect for the entire system (not just for one group or component). A review committee to allocate and review work is feasible as long as the

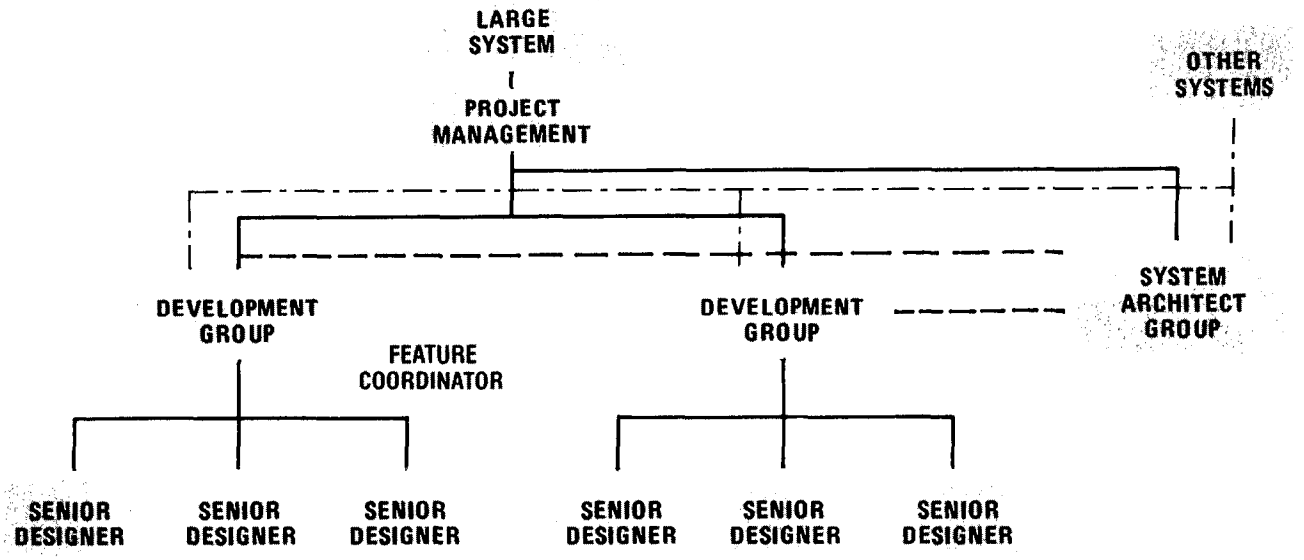


FIGURE 6. The System-Architect Group in a Large System

system is not terribly large (i.e., much beyond two or three development groups).

The organizational recommendations for a large-system development effort are given in Figure 7. System-architect responsibilities are broad in scope overall, but relatively shallow in terms of the development process, whereas feature-coordinator responsibilities are narrow in scope overall, but comprehensive in terms of the development process. Initially, the feature coordinator inputs into the system architect's work, whereas in the later stages the system architect inputs into the feature coordinator's work. Since system-architect responsibilities in a large system cannot be performed for all possible features, topics that are selected for review by the system architect should be either exploratory designs and feasibility analyses, issues with whole-system impacts, or new-development directions.

In the active phase of system design, the system architect reviews his or her design with the feature coordinator and the system designers (the lead relation-

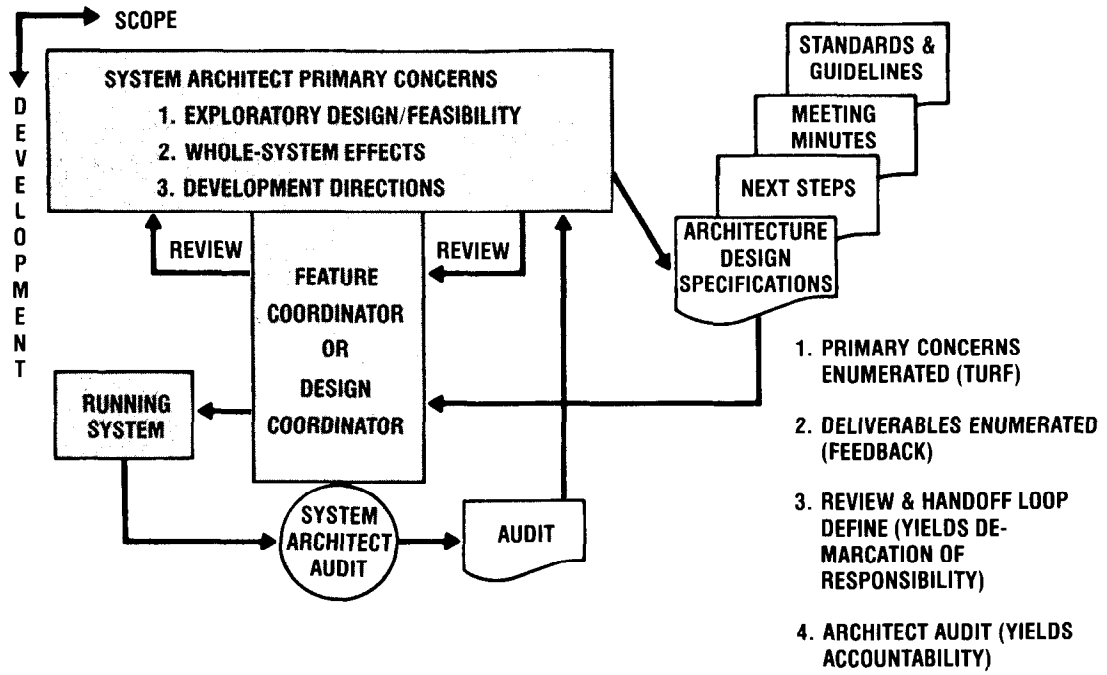


FIGURE 7. Recommendations for a Large System

ship) and produces these deliverables:

- an architecture-design specification—the framework and externals design that initiates the process,
- next steps—the action to be taken by the system designer or feature coordinator, and
- standards and guidelines.

As these deliverables are handed off, the system architect holds architecture-design reviews and tracks unresolved issues in a consulting relationship with system designers.

Once the system is up and running, the system-architect group should audit itself to determine:

1. Was the architecture design fully implemented? Were open issue resolutions implemented?
2. If not, why? What was wrong with the architecture design and/or resolutions?
3. If they were implemented, what problems arose with them?
4. What problems were encountered that the architecture designs or design reviews should have resolved, but did not?

An audit document covering these points should be published, and corrective action taken where necessary.

APPENDIX

A COOKBOOK OF SYSTEM-ARCHITECT RESPONSIBILITIES

In general, the system architect is responsible for the whole-system view, which refers to either the system or part of the system for which his or her development organization is responsible. To provide the whole-system view, the system architect must understand

- "where" the system belongs—he or she steps back and looks at the system as one unit within the greater context of the systems and environment with which it interacts or interfaces;
- "what" the system does—the system architect must be able to enumerate the parts of the design, their functions, the features of the system, and how the parts and features overlap;
- "how" the system acts—he or she understands the data and control flows throughout the system, and how the parts or components cooperate to form a whole.

Besides maintaining the overall view of the system, the system architect is responsible for five areas: *features, designs, interfaces, problems,* and *voids.* As features, interfaces, and designs interact to define each other and generate more features, interfaces, and designs, voids and a variety of problems appear that must be resolved in terms of the system as a whole (Figure 8). To resolve with minimum impact a problem arising in one component, the extent of the problem must be determined, the component where the resolution is primarily implemented pinpointed, and the side effects of that change on all components assessed.

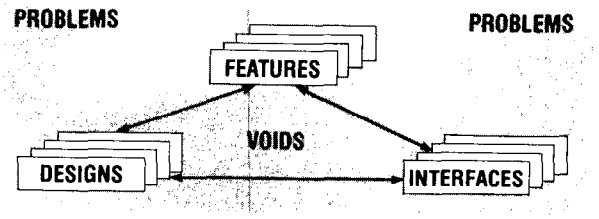


FIGURE 8. System-Architect Responsibilities

The remainder of this Appendix discusses in more detail the specific responsibilities of the system architect. These responsibilities will not be carried out in their full breadth in all cases. In a very large system, for example, only the essentials—ensuring the integrity of the system and its future direction—may be covered.

DESIGN RESPONSIBILITIES

The system architect formulates the definition and interfaces of high-level components, reviews designs, ensures design consistency, and designs future features.

Formulate High-Level Component Definition and Interfaces

Designing the framework and externals of a system involves formulating the definitions and interfaces of high-level components in cooperation with the responsible system designers. (A component is the highest level subsystem of a system.) In a system for which development has already started, the system architect may find the developers working on the most obvious, important, or interesting features; they may even have worked out some ideas on other features. The system architect must now take the core design and the remaining features and group them into high-level components with clearly defined interfaces that may lend themselves to being assigned to individual designers. Figure 9 shows how a system consisting of a core design and features a through g was formulated into five high-level components and their interfaces. Component M3 of this system contains features e and f; it message switches to component M1, which contains the core design and features a through c. M1 message switches to component M4 and calls component M2, and component M5 updates a database that is input to M2.

In this formulation process, the system architect should insist that common routines or functions be used whenever possible. If HELP screens or screen security is needed, are there standardized programs that can be used? If there are not appropriate common routines, the system architect should propose that such standardized functions be developed. Ideally, the entire system should be built up from prefabricated building blocks.

Review Designs

The system architect attends design meetings to review high-level component and lower level module designs. To such meetings, the system architect brings a strong

software-engineering knowledge, an understanding of the context of a component or module in terms of the rest of the system, and consistent review across all components. He or she reviews systems software to ensure consistency of design and proper implementation of interfaces, learn the design, guarantee that all features are implemented, and root out any problems.

Ensure Consistency of Design

Design requirements and methodologies are chosen with a view to making the system robust, easy to develop, enhanceable, and maintainable, while still optimizing performance. Once the methodologies are chosen, the system architect ensures that they are actually used. If a component is to be reusable, the architect reviews designs to ascertain that it is in fact reusable. If a certain interface or protocol is to be used among components, the architect ensures that it is used.

Design Future Features

During the development of one version of a system, future features will be proposed. To determine whether these features are feasible or not and test the robustness of the system, he or she will have to design the new features into the system to the extent that their impact can be determined. The design should show a *proposed* implementation of a feature and its impacts. The design should also show how easily the current system can absorb changes and whether some immediate ones are necessary to increase the system's enhanceability.

INTERFACE RESPONSIBILITIES

The system architect identifies the need for an interface, which may arise while formulating high-level components, designing new features, or resolving problems. The system architect also negotiates the design of interfaces with external systems and mediates the design of component interfaces among designers. This last effort is one of negotiation, rather than design, since the design of the interfaces is greatly determined by the design on either side of the interface, and designers must have considerable input into interface design. This is not to say that the system architect has no input. On the contrary, to negotiate an interface, the system architect must be able to propose alternate designs that will be acceptable to both sides. Interfaces occur

a. between components of a system (internal interfaces),
b. between systems of a project (intraproject interfaces), and
c. between the system and other application systems (external interfaces).

The difficulty of negotiating an interface increases progressively from interface type a to c, whereas the acceptable amount of coupling decreases from interface type a to c. Interfaces in the same system (type a) entail fewer principal people and are more likely to be developed in the same environment using the same philosophy. Interfaces with entirely different application systems (type c) can often be almost intractable. The outside systems may have been developed in entirely different environments and even for different hardware. The development schedules are not likely to coincide, and the development priorities may be at odds. Negotiating interfaces among the systems of a project lies somewhere between these two extremes.

Ensuring that the required data are available and finding an optimal point in the data flow for the interface are the responsibility of the system architect. Since the core of any interface design—the attribute that can make or break an interface as it evolves from version to version—is the amount of coupling between interfacing systems, the system architect must promote the appropriate level of coupling. The more distant two sides of an interface are in their development efforts, the lower their level of coupling. If two components are in the same system and likely to remain there, the coupling is a matter of internal maintenance and modularity. With components in different application systems, coupling is a matter of adjusting to each other's environments and development schedules. In this case, decoupling is desirable: A decoupled implementation will tend to be table and explicit protocol driven.

FEATURE RESPONSIBILITIES

The system architect accounts for all features, forecasts new features, analyzes and proposes new feature de-

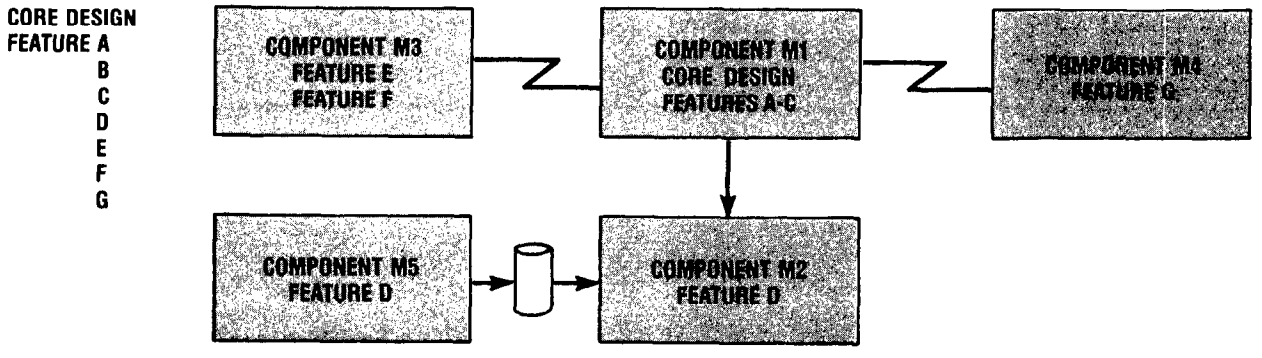


FIGURE 9. Formulate and Unify High-Level Component Definition

signs, and ensures that future features can be implemented. The system architect ensures that all features are included in the design, including auxiliary features like HELP, tutorial screens, and terminal security, and applies his or her knowledge of the application and software technology to propose new architectural features.

Analyze and Propose Design of New Features
The feasibility and impact of a new feature are best determined by proposing a design of the feature. Even though the feature may not be implemented until a later version, everyone will have a "warm feeling" that it is workable.

Ensure that Future Features Can Be Implemented
Once new features are identified and studied, the system architect must keep the design open for both new and unforeseen features. In this capacity, the system architect must have a sense of where the system can go and how far it logically can be extended and still maintain its identity. One way to do this is to design the system as generically as possible.

Problems
Possibly the most critical and visible responsibility is problem solving (i.e., *fire fighting*). The system architect will probably start on a job with a list of problems to which people will add. However, the system architect must be able to locate problems as well as respond to them. Problems will most typically occur in interfaces, system usage, allocation of functions, and the direction of the design (generic, decoupled, top-down, etc.).

The system architect must root around in designs and interfaces to determine the extent of a problem and propose a resolution, isolating the problem and proposing a fix that minimizes the impact on the system as a whole. Problems should be resolved consistently whenever possible: That is, if the solution architecturally belongs in a component, it should be resolved there. The goals of ensuring consistency and minimizing impact will often conflict. In a case where the resolution must be implemented immediately in a critical period of development, it may be more important to minimize the impact than be consistent (e.g., a "quick-and-dirty" fix may be put in the "wrong" component to expedite the solution).

A system architect both *generates* solutions and evaluates solutions from other sources. His or her credibility is predicated on the ability to find solutions that work: If the best solution is available, take it; if one is not forthcoming, invent it! Often the system architect's *ideal* solution, after discussions with designers, proves impractical in the end. The system architect must know when and where to compromise to meet practical needs and still keep the system architecturally sound. To ensure that the agreed solution gets implemented, the mere presence of the system architect in a design review is sometimes sufficient. At other times, it is necessary to return to the rationale of the solution and convince people anew.

Filling In
The system architect fills in when something needs to be done and no one is available to do it. Normally, this is done only until someone else can be assigned. The system architect may also stand in for a person or group of people in a design meeting.

KNOWLEDGE BASE AND ABILITIES

The system architect must have a firm technical background (e.g., in computer science), a thorough understanding of the application being automated, and some knowledge of the external systems with which the system interfaces.

The system architect must understand application requirements, how the system is used by the customer, how the system impacts the customer, the priority of features, and how exogenous events (e.g., a radical shift in the customer's future operations) may affect the application. He or she must have sufficient knowledge of external systems to develop long-lasting, robust interfaces. In particular, a system architect must understand all the external systems that interface or may interface, each external system's effect on the system, the system's effect on each external system, and any problems of the external systems that might affect the interface.

To view a design broadly, stepping out of the ongoing hurly-burly of development, the system architect should be able to see the design as it is currently, as it should be from a good architectural point of view, and, more pragmatically, as it can be given future requirements and resources. He or she should be able to develop a "specific" as well as a "general" solution that solves a class of problems, and then determine which is best. If a problem class cannot be solved by defining specifics in tables, options, and very small logic routines, a specific solution may be preferable. If a problem class can be solved only by either writing elaborate logic for each case or constructing complicated tables, then solving each specific separately may be superior. On the other hand, if the specifics of a problem class can be defined in tables, options, and small logic routines, then not only are a lot of current problems solved, but possibly many future ones as well.

Interacting with people regularly and frequently requires that the system architect be able to get a discussion rolling and keep it on track, listen and consider others' ideas, not antagonize people, and defend and sell solutions to both designers and management. He or she must keep all designers and management informed about problems, solutions, interface developments, and new features.

The system architect must be able to bring appropriate problems to management's attention, tell them when they are wrong, and indicate when they are creating a bottleneck or when they are responsible for the solution of a problem. For example, when schedule tightness is aborting the best solution, it is management's responsibility to decide whether to slip the schedule or implement a less desirable solution. A system architect must be able to present management a proposed design or solution in such a way that the

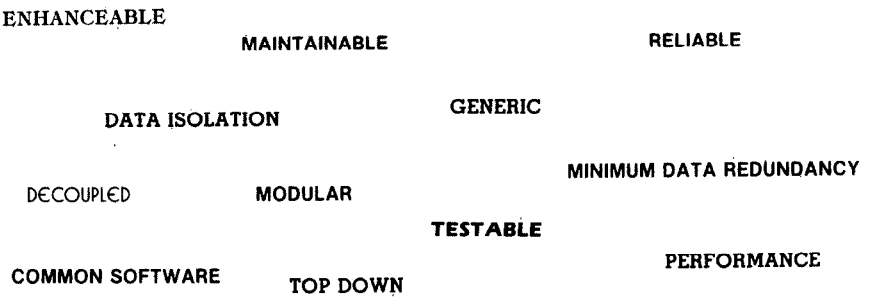


FIGURE 10. Philosophical Buzzwords

effort gets started immediately; this usually involves a statement of the problem and discussion of the solution—how it works and why it was chosen, a list of everyone involved and their responsibilities, and what to do next.

Keeping track of who's who in the various groups is also important: Who knows needed information? Who is ultimately responsible? The system architect must be able to delineate everyone's turf, including his or her own, and not tread on someone else's. He or she must remain open-minded to all solutions, features, and designs; know his or her own limitations; and bow to others' knowledge.

Finally, the system architect needs an intuition for good design architecture, which is often defined by a variety of buzzwords (Figure 10). With proper usage and moderation, the system architect will find the ideas represented by these terms useful. They provide a foundation and framework of concepts that can be used to evaluate designs and solutions, but generally their full application is asymptotic. Trying to adhere to these concepts religiously can lead to schedule disruption and inability to solve problems not accounted for by the concepts.

The set of concepts the author has found most useful in evaluating an architecture is *modularity, top-down decomposition, decoupling*, and *genericness*. Associating components with functions of the application starts the system off with a firm anchor to the application, and then decomposing the functions piecewise makes it progressively more manageable. These processes should be guided toward appropriate decoupling of modules and data, which makes interfaces more manageable and problems more isolable. Designing a system more generically protects the system from unpredicted problems in the sense that, if the generic was developed well, it has implemented a solution to a class of problems rather than just a single problem.

In the spirit of Hofstadter [4], the system architect looks for organizing factors in a system, for patterns and isomorphisms in the features and designs, and for hierarchical, networking, and other connective relationships. The architect must seek out recursive and symmetrical relationships in the system: Patterns allow functions and data to be usefully grouped, isomorphisms identify where one design can serve many features and functions, connective relationships organize dissimilar features and functions, symmetry indicates a balance between the ins and outs and start and finish of a system, and recursion establishes hierarchies of isomorphisms and the interfaces among the various levels. Recursive relationships can be the most powerful of organizing factors, but also the most subtle; they typically require stepping out of the system and looking at it from without. A recursive relationship may be entirely unobvious, but, once found, will be a powerful organizing factor.

In short, the system architect is the whole-system designer, a fire fighter, mediator, and jack-of-all-trades. He or she provides an invaluable function on large-scale systems, bringing worker level, whole-system unity and continuity to a design and offsetting the necessary compartmentalization of modern, modular designs.

REFERENCES

1. Aron, J.D. *The Program Development Process: Part II, the Programming Team*. Addison-Wesley, Reading, Mass., 1983. Establishes the system architect as designer of the framework and externals of a system.
2. Brooks, F.P., Jr. *The Mythical Man-Month—Essays on Software Engineering*. Addison-Wesley, Reading, Mass., 1975. Describes many "real" aspects of developing a large software system, including the idea of system architect.
3. Gunther, R.C. *Management Methodology for Software Product Engineering*. Wiley, New York, 1978. Defines the functions of the chief programmer.
4. Hofstadter, D.R. *Godel, Escher, Bach: An Eternal Golden Braid*. Basic Books, New York, 1979. A philosophical exploration of the relationships among ideas: recursion, isomorphism, perception, etc. Provides food for thought for the system architect.

CR Categories and Subject Descriptors: D.2.9 [**Software Engineering**]: Management; K.6.1 [**Management of Computing and Information Systems**]: Project and People Management; K.6.4 [**Management of Computing and Information Systems**]: System Management

General Terms: Management

Additional Key Words and Phrases: system architect, system designer, development organization, isomorphism, recursive and symmetrical relationships

Received 9/84; accepted 1/85

Author's Present Address: John A. Mills, Bell Communications Research, Inc., 33 Knightsbridge Road, Room 4H-308, Piscataway, NJ 08854.