Software Engineering Process Model Case Study

Gary J. Nutt
Department of Computer Science, CB 430
University of Colorado
Boulder, CO 80309-0430

(303) 492-7581
nutt@cs.colorado.edu

CU-CS-760-94 December 1994

University of Colorado at Boulder

Technical Report CU-CS-760-94
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

Software Engineering Process Model Case Study

Gary J. Nutt
Department of Computer Science, CB 430
University of Colorado
Boulder, CO 80309-0430

(303) 492-7581
nutt@cs.colorado.edu

December 1994

Abstract

Models of processes and procedures are the basis of software to automate parts of office work, business process reengineering analysis and design, general workflow enactment systems. The software engineering discipline has (almost) independently derived a similar need for models of procedures used to specify, design, and implement software products. Software engineering and software processes have risen from the well-known crisis in which very large software projects tend to solve a different problem than the one intended. The government has published a standard (military standard 2167A) to describe documentation to ensure that software can be specified and tested by independent contractors. This paper is a case study of how a government software contractor might use models to define a process for designing and implementing a software product that complies with the documentation requirements. The intent of the paper is to apply group process modeling technology to the software engineering domain, thus exploring strengths and weaknesses of our evolving models of group collaboration. The case study illustrates an alternative way to design, analyze, and track software processes. It also attempts to illustrate how the would model "break down" as the basis of an enactment model if it were to be used to coordinate the work of a large number of software developers. Therefore the paper implicitly suggests new directions for workflow model and system development and research.

1 Introduction

1.1 Process Modeling

Models have been used to describe work processes by decomposing the procedure into a set of discrete steps, then showing how information flows among them. The purpose of the process is arbitrary, and has traditionally varied from manufacturing processes to information management strategies. Variants of these process models have been heavily-used for modeling specific behavioral aspects of organizations: PERT charts describe how manufacturing and engineering processes can be organized to build a product. Queueing networks model workflow through a system of service providers in terms of service and interarrival times. Flowcharts describe how a sequential program should execute to transform information.

A *workflow language* is a mechanism by which the process steps are explicitly identified and the description of how work flows among the various steps is defined. The nature of the language for expressing the workflow approach depends on how the workflow specification is intended to be used:

1. The language may be used exclusively to document a procedure so that it can be understood by everyone that has an interest in that procedure; this is a typical use of PERT charts.
2. It may be used to specify the procedure's behavior under different loading conditions with varying service rates, allowing one to predict the performance of the procedure; this is how queueing networks are used.
3. The language may be used to precisely define a set of actions that must be conducted for the procedure to work; this is typical of a graph-based visual programming language (and state diagrams).
4. Thus workflow representations range from informal, *descriptive* representations to formal, *prescriptive* representations.

Some workflow languages may have an underlying formal model that define their syntax and semantics (i.e., their behavior); Petri nets [16] are one such formal model, and ICNs [5] are another. *Uninterpreted modeling languages* focus on describing the flow of work in the model and the *coordination* among step execution, but ignore the details of processing within a step. The semantics for more detailed workflow models are represented by an *interpreted* formal model such as high level Petri (predicate/transition) nets [11, 14] or ICNs with activity interpretations. The amount and type of detail specification reflects on how the workflow language will be used.

Representing Procedures. Organizations operate on a set of basic principles and procedures. In some organizations there are extensive procedure manuals that describe how operations can be accomplished within the enterprise. Other organizations do not bother to write procedure manuals since it is difficult to express the procedures in a manner that can be understood by all of the employees. Secondly, the procedures tend to change as the corporation gains knowledge about the procedures or as the operation of the business evolves. Third, the specification may be too prescriptive for human workers to follow and still be effective.

Thus representing procedures is an important application for workflow tools. A good workflow system can provide a dynamic means for documenting the steps and their interrelationships for many procedures in the enterprise. (Not all work can be represented well with workflow; the work steps that cannot be represented may actually be more *principles* than procedures.) The workflow specification can be quite detailed if the enterprise intends that there be little deviation in the way that the procedure is to be executed (e.g., if the procedure must satisfy some external specification of the steps and the order of their execution); conversely, it can be quite vague if the details of the steps are unknown a priori, or if the details of step implementation are really unimportant with respect to the overall organization. The level of representation reflects the concern of the procedure designer with how procedures are to be carried out in the enterprise.

Computer technology can substantially change the way one represents procedures. With the extensive realtime graphics facilities available, one can provide a broad spectrum of dynamic as well as traditional static representations of work. Electronic representations have the advantage of being easily edited, updated, and distributed. This has always been a severe limitation of paper procedure manuals. Second, electronic representations can be interactive, allowing the user to navigate through the procedure specification using database and/or virtual reality technology. Third, electronic representations can incorporate animation facilities to provide more complete representations than are possible on paper. Fourth, electronic representations can incorporate multimedia, allowing the procedure to be described using text, graphics, images, and audio media. Thus, we view electronic descriptive workflow systems as an important aspect of workflow applications, albeit technically trivial compared to enactment applications.

Descriptive Workflow Models. A workflow model that focuses on documenting and describing processes must provide facilities for identifying each step in the procedure, then showing how the execution of different steps are related. The model will be produced by humans (using computer tools) to communicate ideas to other humans. Therefore the modeling language and support tools are required to be flexible and concise in term of their ability to describe a procedure without being concerned with detailed descriptions of steps.

For example, in the *Information Control Net* (ICN) model, steps in a procedure are called *activities* [5, 6, 8].[1] *Control nodes* are added to the language to specify that work may flow to (from) alternate activities when a predecessor activity has been executed — called disjunctive (exclusive OR) logic. ICNs also allow control flow to divide so that two or more activities can be executed concurrently (or in any order) — called conjunctive (AND) output control flow. Conjunctive input control flow is used to show that an activity can only be executed when all of its predecessor activities have completed — called AND input control. Pictorially, activities are represented by large, open circles, OR logic by small, open circles, and AND logic by small, filled circles.

The uninterpreted control flow can represent execution by showing how work flows through the net. This is accomplished by representing units of work as tokens that flow through the ICN graph, i.e., the net represents the steps and the flow rules, while a token can be placed on a node to represent that work is being done at that node. When the token passes through an AND node with multiple output arcs, the token is cloned and replicas are placed on each output arc; when a token is present on each input arc of an AND node with multiple input arcs, the node will fire, placing one token on its output arc. Thus token flow explicitly defines the way that work flows through the uninterpreted control flow net. In workflow terminology, the token is called a *workcase*.

Empirical experience with ICNs suggested that the representation powers of the model could be much greater if the uninterpreted control flow net also represented how activities read and write various data repositories in the system. Let a square represent a data repository with an arc from a data repository to (from) a control node representing that the activity may read (write) data from (to) the data repository.

Basic ICNs model office work, independent of the assignment of the work to positions or individuals (including computers) in an organization. They do not explicitly take resource utilization into account; rather, the analyst is expected to model specific resources as they effect the procedure. This can be accomplished by explicitly describing when a specific agent executes an activity. A consequence of this approach is that tokens may represent both workcases and agents (called *actors* in the workflow literature). While the approach is representationally complete, it has three significant negative aspects: it complicates the expression of the procedure, second it makes it difficult

[1]ICNs share many properties with the Process Interchange Format [15] and the Workflow Management Coalition model [21].

to assign a single actor to multiple roles, and third it does not provide any obvious mechanism to represent actor scheduling/preemption (to multiplex across pending work).

Actors may be people or computers, and their *roles* identify a set of activities that the actor is capable of performing in sequence (i.e, an actor represents one "processing entity," so it can only do one thing at a time). In extended ICNs, tokens represent only transactions (workcases) while all processing agents are represented with the role and actor constructs.

The role identifies a particular type of processing, e.g., a purchasing agent, but it does not identify the person or computer that can/will execute the role; that is done by explicitly *scheduling* an actor to a role for a workcase. Thus the model must associate (map) actors with roles, meaning that any actor is capable of playing the role to which it is associated. When a workcase needs to be processed, then an actor that is capable of playing the role is assigned to the workcase while it is in the activities associated with that role. It is also possible that any given human actor can hold many different roles (time multiplexing across those roles), such as purchasing agent, supervisor, employee, etc.; so this mapping is a many-to-many mapping.

Analytic Workflow Models. Once workflow models are supported by computers, an obvious extension is for the system to also perform various analyses; therefore most computer supported workflow systems are also analytic workflow systems.

The technical part of the analysis problem is heavily dependent on tools. As the procedure architect considers different alternatives, he needs to be able to describe the architecture in relatively precise (although not necessarily detailed) terms, then to experiment with the model to understand its behavior under different loading conditions. Perhaps the simplest form of analytic support is to *animate* the token/workcase flow through the model; such a tool does not necessarily provide quantitative measures of the performance, but it can give the analyst a valuable *qualitative* understanding of how the system will react to different loading conditions. For example, if the flow of the work through the system is subject to bottlenecks, then experimentation with the animation will often illuminate these areas with very little experimentation. Animation has become a fundamental "first cut" analysis tool for complex systems.

Qualitative observations of the performance of the system require more information in the model, more experimentation with the model, and more sophisticated support from the system. In some cases, the workflow system can be analyzed like a queueing network; this may result in closed form mathematical expressions to define performance measures.

Like animation, simulation requires that the model be interpreted. However, in simulation, the qualitative results are presented as post execution reports. Therefore, model interpretation need not be done in scaled realtime (in fact, scaled realtime is usually an annoyance when one wants the qualitative reports on system performance. Such reports will normally address throughput, turnaround/response time, availability, and resource utilization. In a workflow model, resources are represented by actors, so the resource utilization statistics must correspond to the way actors do their work (within the role framework) under varying workcase loading conditions. The load on the system is workcases, so part of the reports will be related to characteristics of how quickly workcases get processed, how much delay they encounter at various scheduling points, etc. It is also worth noting that a workflow system probably cannot "automatically" produce the desired performance reports without some specification of *which* resources are of interest with respect to workcase flow; this suggests that the workflow system incorporates facilities for selective report generation (this concern is common in performance measurement domains).

The uninterpreted ICN model can be refined so that it is a model suitable for analysis. To accomplish this, one must add interpretations to the executable entities in the model to represent their individual performance characteristics. The complete workflow model will use these individual performance characteristics (e.g., a description of the amount of time a particular activity takes to execute on a particular type of workcase with a particular actor) to determine the overall performance characteristics of the model, then report the observations as specified by the analyst.

Roughly speaking, the performance behavior of a node (activity, control node, or data repository) is represented by specifying how long it should take for the activity to execute whenever the model semantics require execution. The simplest such characterization is to use an average execution time; the simulator moves tokens representing workcases from activity to activity as determined by the topography of the network, leaving the token on the node for a simulated time corresponding to the execution time characterization. The aggregate behavior of the system — actor utilization and workcase flow characteristics — is represented by the way tokens move through the graph model.

The model can have more sophisticated interpretations than mean value estimates of the execution time, including probability distribution functions, function subprograms, and verbal descriptions. If the model is to be used for simulation analysis of the system, the interpretation will normally be either a probability distribution function or a function subprogram.

Analytic workflow tools are substantially more complex than descriptive workflow tools, since they must contain much of the functionality of the latter in addition to analysis tools. Most tools are capable of animating and simulating workflow via a point-and-select interface, although the integrity of the underlying models is often suspect. Many systems provide the capability to specify probability distribution function interpretations, but far fewer allow the analyst to write function subprograms to define the behavior of a node or an output arc selector.

The Quinault ICN system [18] supported probability distribution function node interpretations for ICNs and the Olympus ICN system [17] supports both probability distribution function and C compatible function node interpretations for simulating workflow. These systems use basic ICNs to provide animation and simulation of the workflow (i.e., they do not model actors and roles).

As in descriptive workflow systems, the human-computer interface is critical for initial acceptance. As the systems become better understood by the user, the modeling functionality will begin to dominate the tools success (since that is where the value of such a product ultimately lies).

1.2 Software Product Organization

We are interested in applying workflow modeling techniques to describe the process of creating software products. There is considerable research activity based on the idea that one can represent the steps in producing software as an algorithmic process [12, 13]. This research tends to cast *software process models* as programs in high level languages; we attempt to apply our workflow process models to the software development process.

The software crisis has long been known to be a critical technical problem, and has even been identified as a grand challenge problem [19]. Very large software systems tend to be extremely difficult to manage for a number of reasons: it is difficult to create precise requirements for desired systems. There are few good ways to partition designs and to integrate the resulting modules. There are few good tools for managing the work. Organizations tend to dramatically underestimate the complexity of the target system (workers build far more complex units than the project managers can fathom). Despite years of research in the area, there is no proven methodology for software engineering.

The federal government contracts substantial software projects to independent contractors. Historically, the government has oftened been disappointed by the quality and nature of software that is delivered in response to a contractual commitment. Over the years, this has caused many government agencies to attempt to provide guidance regarding the process that is used to produce the software (without actually prescribing an operational software methodology) so that they can have some assurance that software does what was intended. The military specification 2167A is one standard that agencies may use to constrain the way that software is developed under a government contract [20].

The 2167A specification focuses on documentation rather than on a software engineering development process; however, any organization that intends to meet the documentation requirements will need to employ some relatively confined process to be able to satisfy the requirement. As a practical matter, the standard implies some family of processes under which acceptable software can be developed.

This paper explores the use of process/workflow models to represent the software process for organizations that comply with the 2167A specification; the models are first used to consider *idealized* processes, then to consider more realistic operation.

Under ordinary usage of the 2167A specification, there is a *granting agency* that desires to have some software product built, and one or more *bidding agencies* that could do the work. The granting agency announces a *request for proposals* (RFP) inviting bidding agencies to submit a formal *proposal* indicating how it would produce the software (see Figure 1). The granting agency selects one bidding agency, and negotiates an agreement based on the proposal. The result of successful negotiation is that the granting and bidding agencies agree on the *statement of work* (SOW) that specifies the precise deliverables and a budget. The agencies then commit the statement of work and budget to a contract with specific terms and conditions.

The bidding agency can then begin to do the work under the constraint that its work (demonstrably) conform with the documentation required by the 2167A standard. First, the deliverables listed in the SOW are converted to a *system requirement specification* that identifies explicit, testable requirements for each deliverable. The standard does not allow requirements that cannot be tested, otherwise it is not possible to determine if a deliverable satisfies the negotiated contract or not. For example, a requirement such as "the system shall be fast enough to avoid catastrophe" is not acceptable, while a requirement such as "the response to any command must be accomplished in 11 seconds" is acceptable.

The systems requirement document is used to drive the design; the design phase must produce a *system design document* and a *system test plan.* The system design document must specify:

External interfaces. The specification of all input operations and data for the system, and the corresponding result at all output ports for the system. Each individual requirement in the systems requirement specification must be satisfied by some aspect of the external interface.

Architecture. Describes how the system will accomplish the functionality required of the system. Functions at the external interface must correlate with specific parts of the architecture.

Subsystems. The architecture can be partitioned into subsystems. Required function implementations must be traceable to a subsystem.

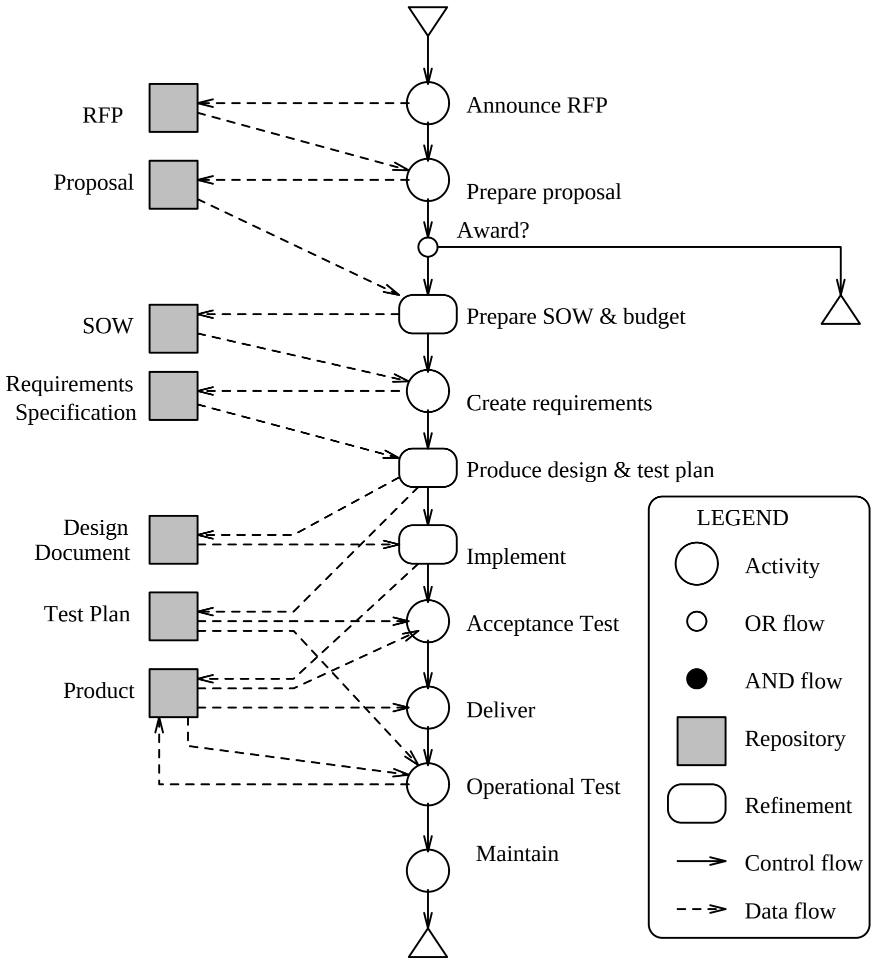

Figure 1: The Waterfall Model (Idealized 2167A Process)

Internal interfaces. The specification of the interfaces among subsystems. Each internal interface specifies an external interface for a subsystem.

The system test plan is prepared at the same time as the system design document, and specifies:

Test Plan. How each unit in the design will be tested to show that it implements specific requirements.

Test Description. The nature of the tests that can be applied to a design unit after it has been implemented to illustrate that it performs as required.

Test Report. A document that is produced when the test has been applied to the unit.

Part of the overall system test plan defines an *acceptance test plan* used at the time the product is delivered, and an *operational test plan* used at the time the product is installed (often the acceptance test plan and the operational test plan are the same).

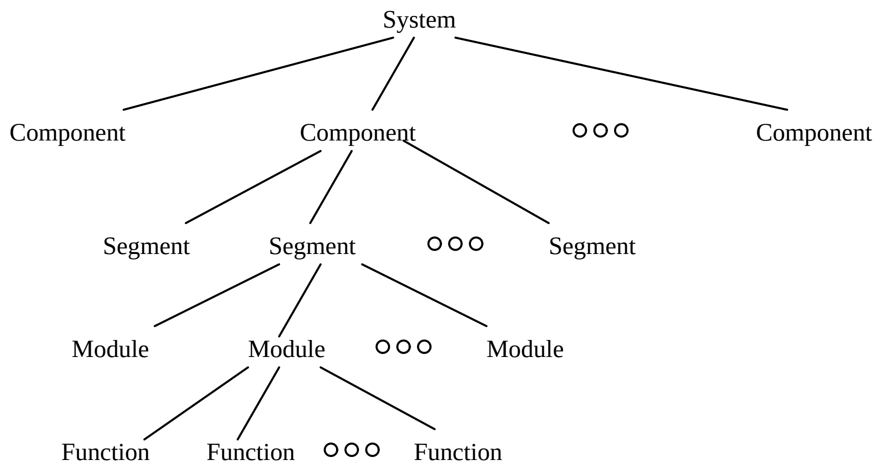

Figure 2: Hierarchical System Organization

Because of the wide use of hierarchical decomposition in systems (software and otherwise) built by humans (e.g., see [3]), the 2167A standard presumes the system will be organized into a hierarchy as shown in Figure 2. For example, a *system* might be partitioned into a set of *components*; each component might be partitioned into a set of *segments*; each segment might be partitioned into a set of *modules*; and each module might be partitioned into a set of *functions*. Design documents and a test plan must be produced for every unit in the system, i.e., when a system is decomposed into a set of components, each component must have a design document and a test plan as described above for the system. In particular, subsystem requirements are derived from supersystem requirements, and the design document and test plan are derived from the subsystem requirements. Each function in the supersystem requirement must have a counterpart in the union of its decomposed subsystem requirements, and into the design document and test plan. Further, most organizations strongly discourage "vertical iteration" in which subsystem design causes supersystem requirements and/or test plans to be changed; otherwise, individual groups cannot be delegated work without the danger of wasted effort, and at the top level this would imply that negotiated agreements with the granting agency might have to change.

Once the system has been completely designed it can be implemented. Implementation is accomplished by delegating work to different groups and allowing them to construct their assigned units of the end product. As each unit is completed, it is tested to ensure that it meets the test plan and hence the cascaded requirements. Integration is implicitly part of hierarchical development.

Ultimately, the full system is built and can be prepared for delivery to the granting agency. Acceptance tests are run at the time the system is released by the bidding agency and initially accepted by the granting agency; subsequent operational testing may take place when the product is installed in its target environment.

Figure 3 summarizes the documents, information flow among documents, and the relationship between the product and the documents. We use a dotted line to represent the hierarchical relationship of documents, i.e., the dotted line from requirement specification to requirement specification is intended to mean that the supersystem requirement specification is used to define the subsystem requirement specification. In particular, we use a dotted rectangle to surround part of the document flow, and a "recursive call" on this part of the process within the flow diagram.

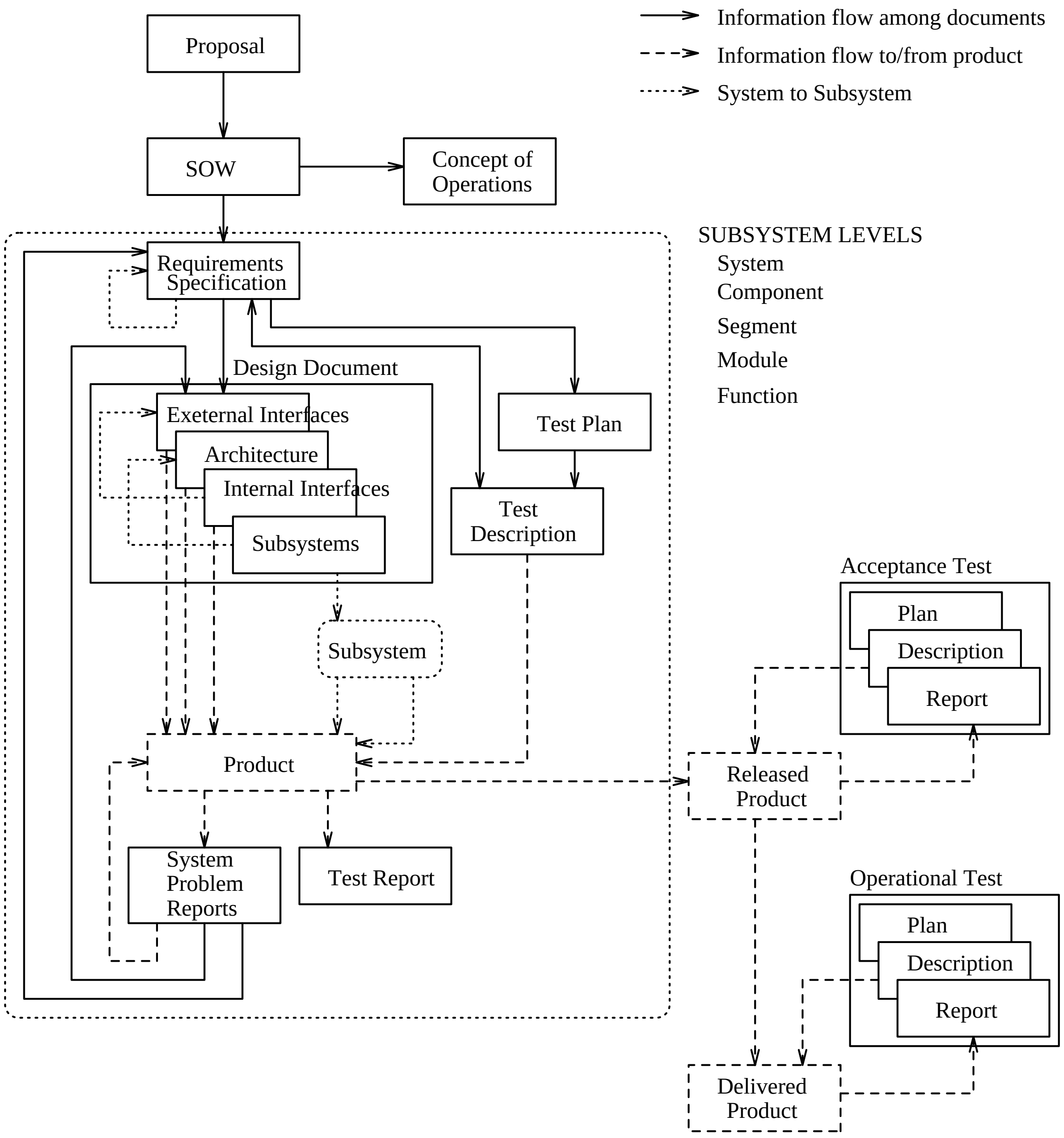

Figure 3: 2167A Document Dependencies

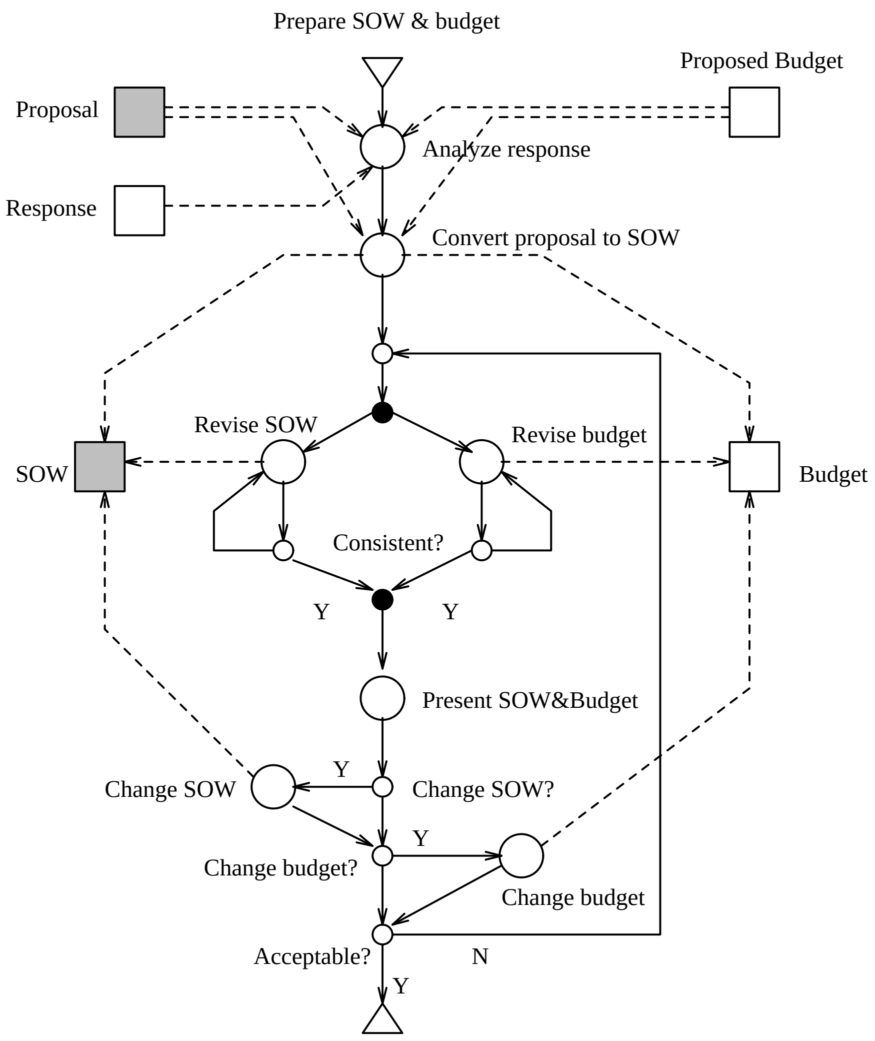

Figure 4: Preparing the Statement of Work (Idealized 2167A Process)

2 An Extrapolated Process Model

Figure 1 is a very high-level view of the idealized (waterfall) process that a bidding agency might choose to employ. Subsequent figures refine the idealized model to provide more detail and alternate perspectives of the process an organization might use to produce software in a manner that satisfies the 2167A requirements.

The SOW is created through negotiation between the granting and bidding agencies (Figure 4). The proposal specifies the basic boundaries of the SOW under a specific budget; it is used as a starting point in negotiations between the two agencies. The response to the proposal is to add some functions to those originally proposed and to remove other proposed functions; the budget is adjusted to reflect the revised SOW. The model describes how a bidding agency might respond to the proposal; it must use the proposal, budget, and the response from the granting agency to create a specific SOW. In the figure, the activity entitled "Convert requirement to SOW"

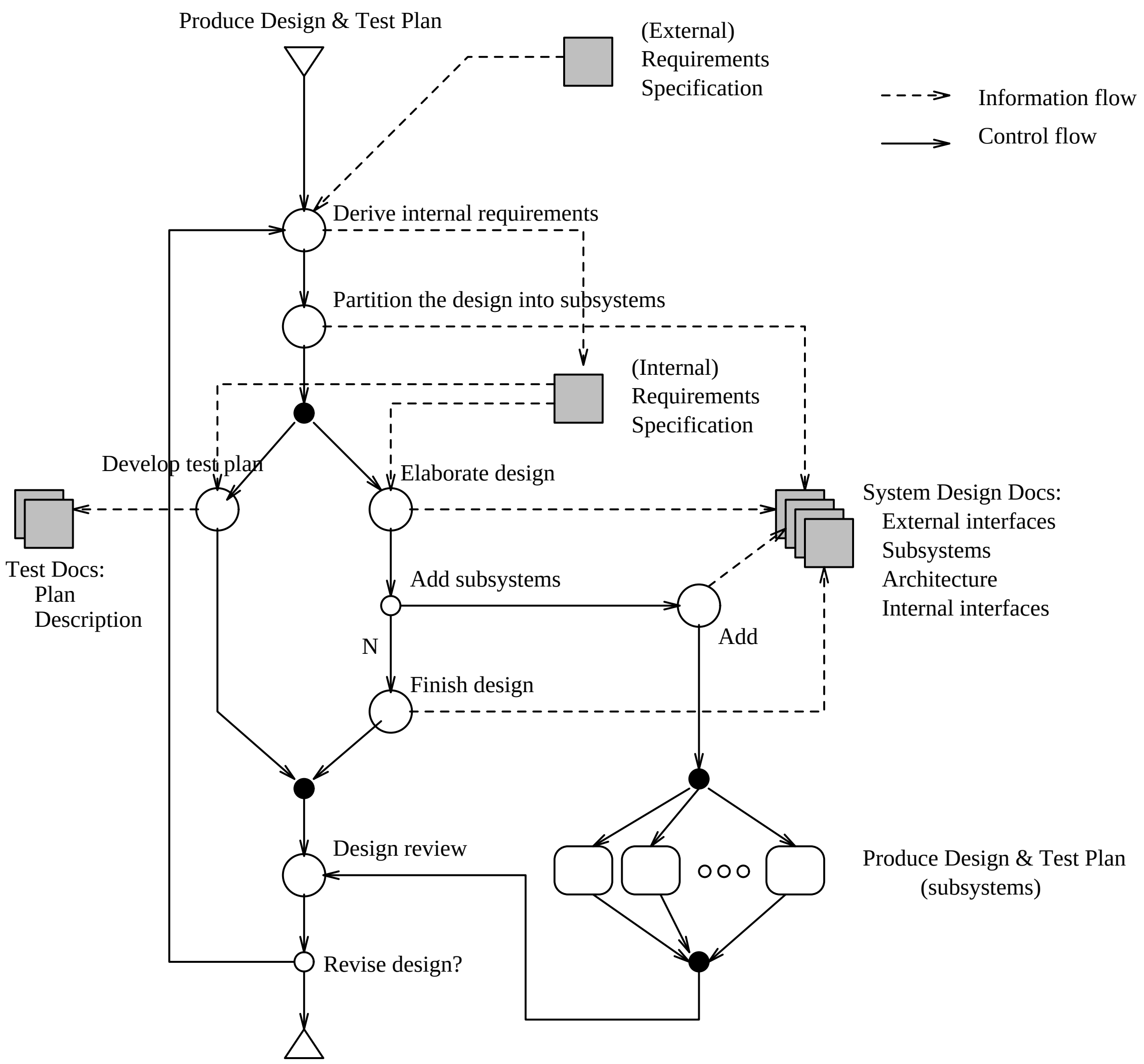
Produce Design & Test Plan
(External) Requirements Specification
Information flow
Control flow
Derive internal requirements
Partition the design into subsystems
(Internal) Requirements Specification
Develop test plan
Elaborate design
System Design Docs: External interfaces Subsystems Architecture Internal interfaces
Test Docs: Plan Description
Add subsystems
Add
N
Finish design
Design review
Produce Design & Test Plan (subsystems)
Revise design?

Figure 5: Produce the Design (Idealized 2167A Process)

represents this work. The SOW and budget are then revised (logically in parallel); this revision process continues until the proposed SOW and budget are consistent. The revised SOW and budget are then presented to the granting agency for approval. The granting agency may accept the SOW and budget or may ask for either to be revised — see the figure. If revision is required, then the process loops back through the part of the process that ensures that the SOW and budget are consistent prior to presenting them to the granting agency again. This iteration continues until the SOW and budget have converged onto an acceptable plan. The bidding agency is then ready to create the system level requirement by translating the SOW into requirements (we do not refine this part of the process).

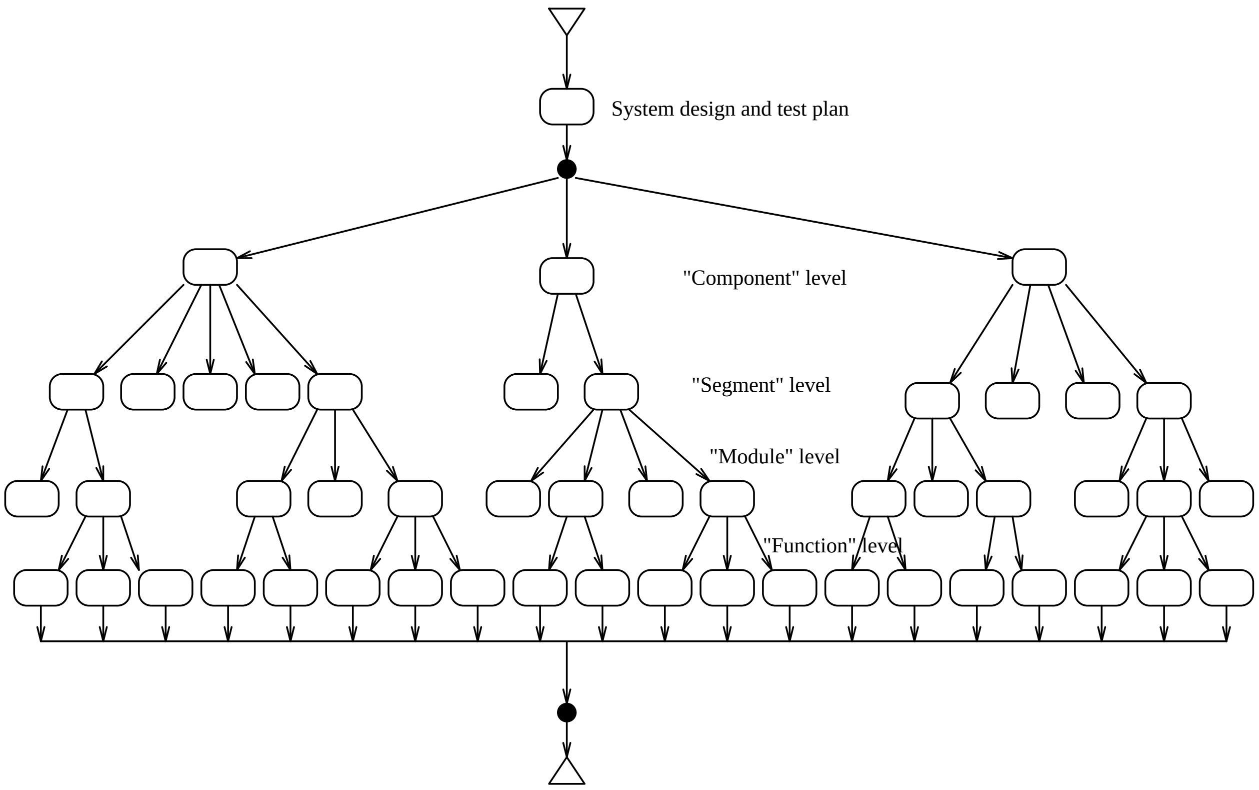

Figure 6: An Instance of the Design Hierarchy

2.1 The Design Activity

Once the system requirements specification has been completed, the system-level design and test plans can be completed. Figure 5 describes more details of this process. This macro step represents a substantial fraction of the work — sometimes *all* of the work on a contract. The model is intended to describe the recursive topdown design process where any system can be decomposed into a set of subsystems: the first step is to create the test plan and design document for the overall system. Part of the work of determining the system design is to partition it into subsystems that can be designed independently, interacting across internal interfaces specified as part of the design at this level. Notice that the model illustrates that subsystem refinement is a *wholly nested* activity, meaning that the system design activity has a single entry and exit point with refinement resulting in path fanout and fanin contained within the internal details, but not visible at the activity interfaces. This property corresponds to the information in the waterfall model, i.e., all design is completed before any implementation is started; it also represents that subsystem design cannot effect supersystem requirements and tests plans. (However it rarely represents the way projects are actually accomplished.)

Bidding contractors approach the project using a process based on the waterfall model with the design elaboration represented by Figure 5. The maximum design depth is likely to be about five levels as suggested by the common subsystems level names (system, component, segment, module, and function), but the common usage has no implied limit or suggestion regarding the breadth. The standard and the model both support arbitrary breadth and depth.

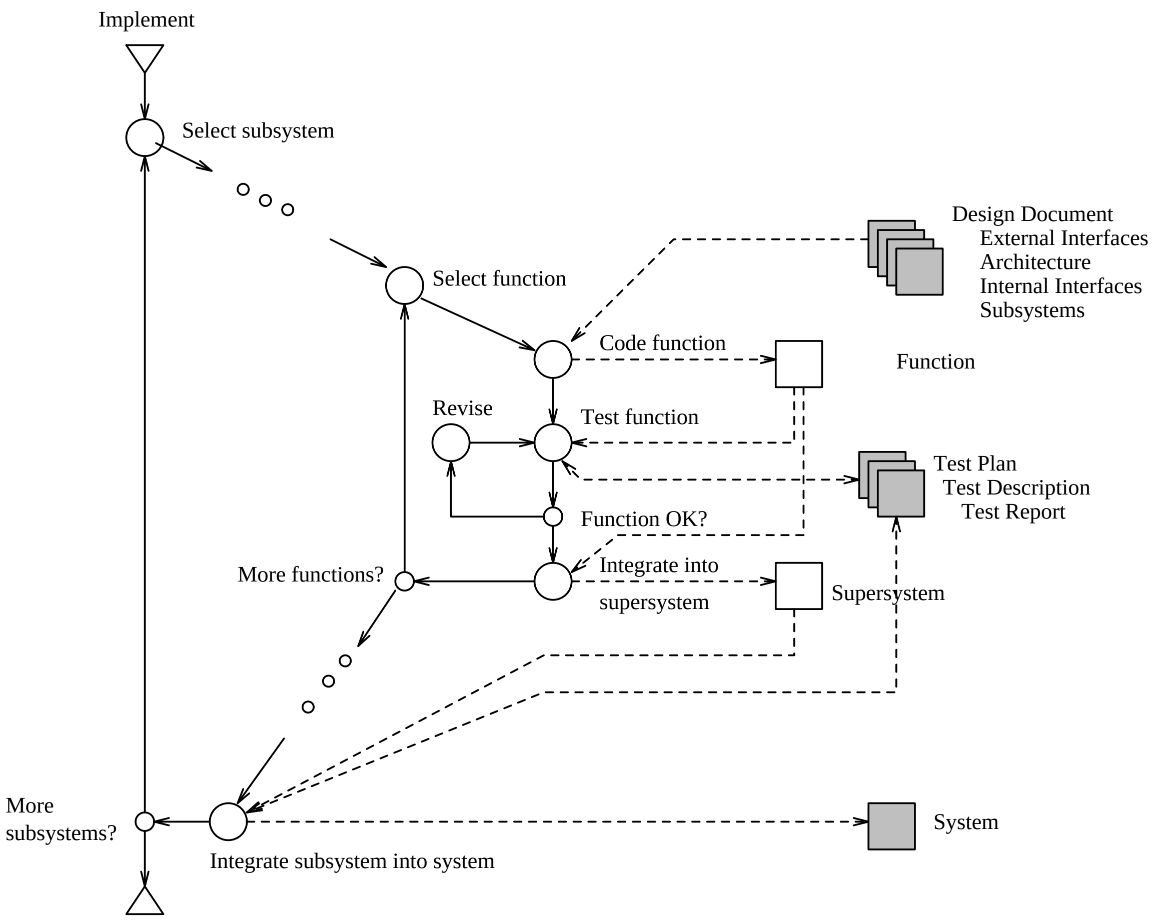

Figure 7: Implementation (Idealized 2167A Process)

Figure 5 is a compact representation of the design step that represents a hierarchy of arbitrary breadth by eliding first generation subsystems, and of arbitrary depth by recursively referencing the "Produce design & test plan" submodel. When the model is used to represent any specific process instance, it results in a tree of design processes as suggested by Figure 6.

2.2 An Idealized Implementation Model

Figure 7 represents a refinement of the "Implement" activity in Figure 1. The model again uses the ellipsis operator to represent depth refinement of the subsystem hierarchy, but models the breadth at any level using iteration. This follows since the design hierarchy results in a set of leaves, each of which should result in the creation of some unit of software — called a "function" in the figure. Functions are encoded (based on the function design), tested to see that they meet the interface requirements and the test plan; a (unit) test report is produced to verify that the function meets the requirements. (Ultimately, each unit test should be traceable back to a high level system requirement.) If the implementation does not pass the test, a system problem report entry is created (not shown in the figure) and the function is refined and retested. Once a function has successfully passed the unit test, it is integrated into its encompassing supersystem — a subsystem

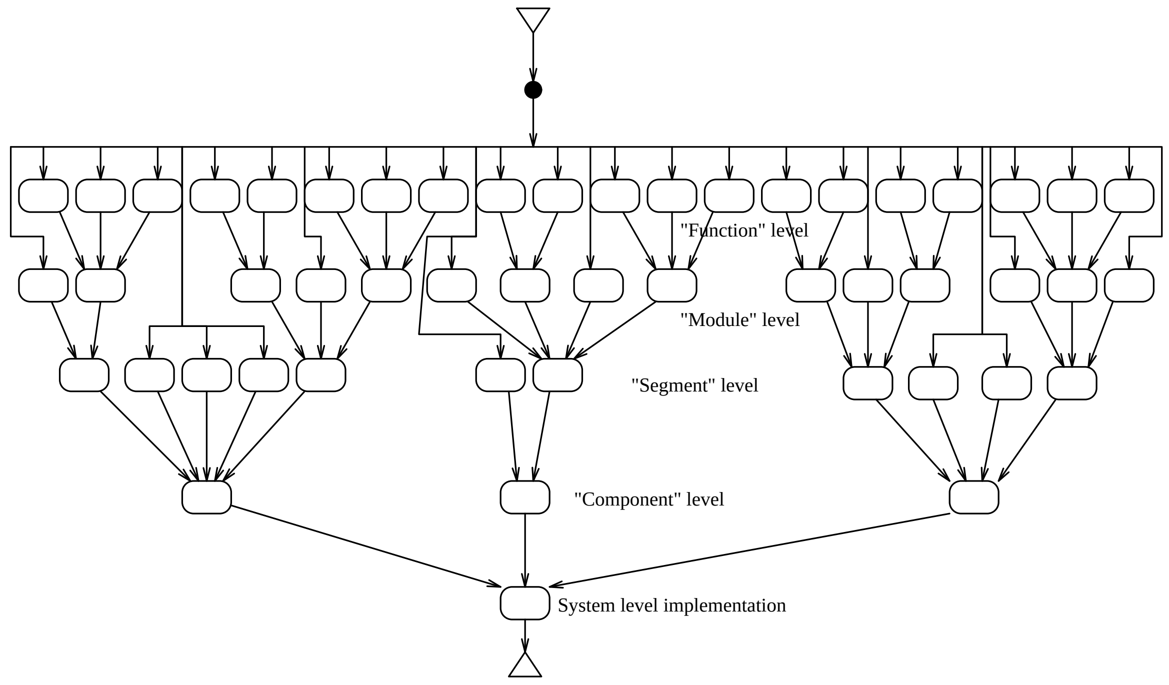

Figure 8: Implementation (Idealized 2167A Process)

in the design hierarchy. In the figure we have elided the various subsystem levels, representing subsystem testing and integration similar to that shown for function integration.

Just as Figure 6 represents an instance of the process for the "Produce design & test plan" activity, Figure 8 represents a hypothetical instance of the "Implement" activity shown in Figure 7. Notice that the synchronization point that represents the completion of the design process enables function-level implementations, module-level implementations (that are not elaborated to the function level), and segment-level implementations. The activities in the figure represent the implementation and integration.

3 Flaws in the Modeling Process

The model described in the previous section represents an idealized view of the software process. It is likely to be useful as a high-level description to explain the process to the granting agency or a new employee. However, it probably is not an accurate reflection of how the software would be created in the organization.

The first problem with the model is that the abstraction of the design and implementation hierarchies is too abstract for practical use. Before it really imparts the nature of a specific process instance, it is necessary to translate the subsystem elaboration into a model that represents the actual design hierarchy such as suggested by Figures 6 and 8. This suggests that if a model and system were to be useful to software organizations, the basic process could be defined using the system, then elaborated for each specific contract.

This representation also suggests a second problem; there are many tasks to perform in the process, but nothing to indicate how the work is delegated and accomplished. Our discussions with

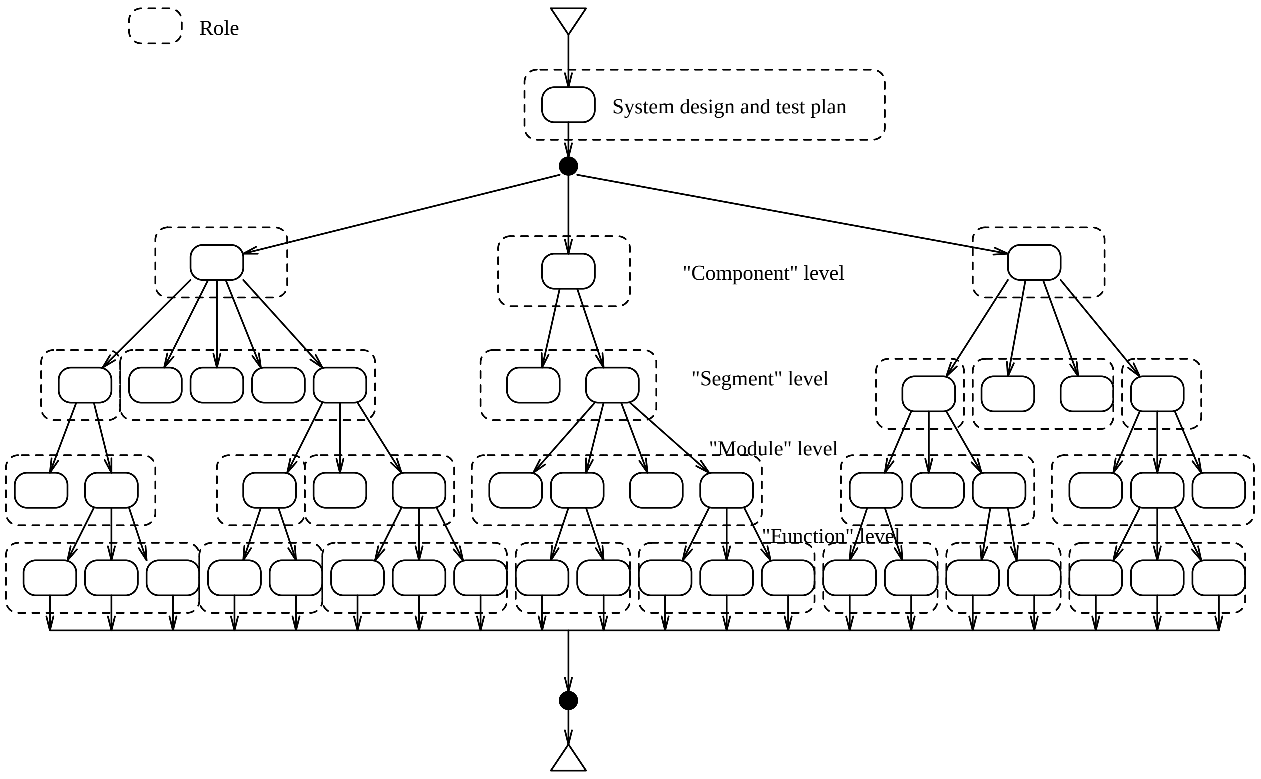

Figure 9: Partitioning the Work among Roles

practicing software engineers is that the models represent the concept, but that they do not capture the essence of the enactment of activities. For example, if a “technical” person is responsible for the project, then various activities are deemphasized while others become the subject of intense attention; conversely, if a “manager” is responsible for the project, then different activities become the focus of activity — substantially different work is done in one case then in the other.

The third problem is the most serious problem (and it is easily recognizable from case studies of office systems): the idealized model represents the general idea, but it does not represent the way work really gets done. This is not due so much to change and exceptions as it is to the fact that the model simply represents the intent of the process rather than the mechanics of the process.

3.1 Roles and Actors

The idealized model can be refined to specify more detail about the process, although the level of description provided in Section 2 may be adequate to address the various documents that are required by the 2167A specification. Each workflow activity represents some unit of work than must be addressed before the overall task is completed. In a model that describes the work of a group or organization, such as a software development organization, the model can be used to represent the assignment of work to various entities. A *role* designates a unit of work — one or more activities — that is to be fulfilled by an *actor*. Thus there is a mapping between activities and roles, e.g., system design activities are to be done by a “senior designer,” coding a function is to be done by a “junior programmer,” and subsystem testing is to be done by a “quality assurance engineer." Conventionally this correspondence maps many attributes to a single role, although there are arguments for allowing the mapping to be many-to-many. There is another mapping between roles and actors that specifies which specific person (or computer) is responsible for executing any particular role; of course there may be many actors assigned to any role, and an actor may have many different roles.

Figure 9 describes a possible mapping of activities to roles. The nature of the organization begins to be evident in a role mapping, e.g., in some organizations it would be unusual for a programmer (or programming group) to be mapped into functions from two different modules managed by two different actors.

The actor mapping will also be determined by the group organization; while roles may be homogeneous across parts of the design hierarchy, work assignment (actor assignment to roles, and hence to activities) would correspond to management responsibility in a traditional hierarchical organization. If the organization used a pool of programmers that could be assigned to any project, then the actor mapping would use an entirely different philosophy.

The model indicates that certain documents are to be prepared (depending on the design decomposition), in a particular order, e.g., the test plan is supposed to be derived from the requirements rather than from the design. There is a "super role" associated with managing the whole process; if the actor that fills this role is concerned with the quality of the documentation, then he or she will tend to closely manage the document preparation activities, but perhaps pay less attention to the nature of the activities that effect the actual design, e.g., the subsystem refinement at any given level. Further, some actors may follow the activity precedence religiously, while others may allow activities to be enacted in orders other than that specified by the model. (Of course out-of-order enactment can easily lead to violations of *intended precedence* in the model, e.g., the test plan development and design document should be concurrent and independent — if the test plan is designed after the design document, then it will tend to be derived from the design and the requirements rather than just from the requirements.)

The way that a role is fulfilled also relates to where it exists in the model. For example Figure 6 makes no distinction between the level of documentation related to system level design and function level design, while in fact, functions may be implemented with very brief (or no) design document and test plan, relying on the supersystem to cover these aspects of documentation. (For example, an external policy could not necessarily distinguish between a part of the design hierarchy in which a module is a leaf and one which is further decomposed into functions.)

While there is a tendency to characterize these differences in actor behavior by the occupation of the actor (manager or technical person), it is also influenced by the nature of the group (e.g., is it market-driven or technology-driven), the size of the group (e.g., large groups need to have more rigorous rules and adherence to rules than do small groups).

3.2 Enactment versus Intent

Figure 10 combines Figures 6 and 8; it is included to point out an obvious failure of the model when it is considered in detail. In this process *no* part of the implementation can proceed until *all* parts of the design are complete. In a realistic project, many aspects may not be known until far into the project; this will tend to cause gross inefficiencies in the organization — workers will remain idle waiting for their part of the design (which is complete) to be released until all other (related and unrelated) parts are all completed. Organizations rarely behave in this manner; instead, parts of

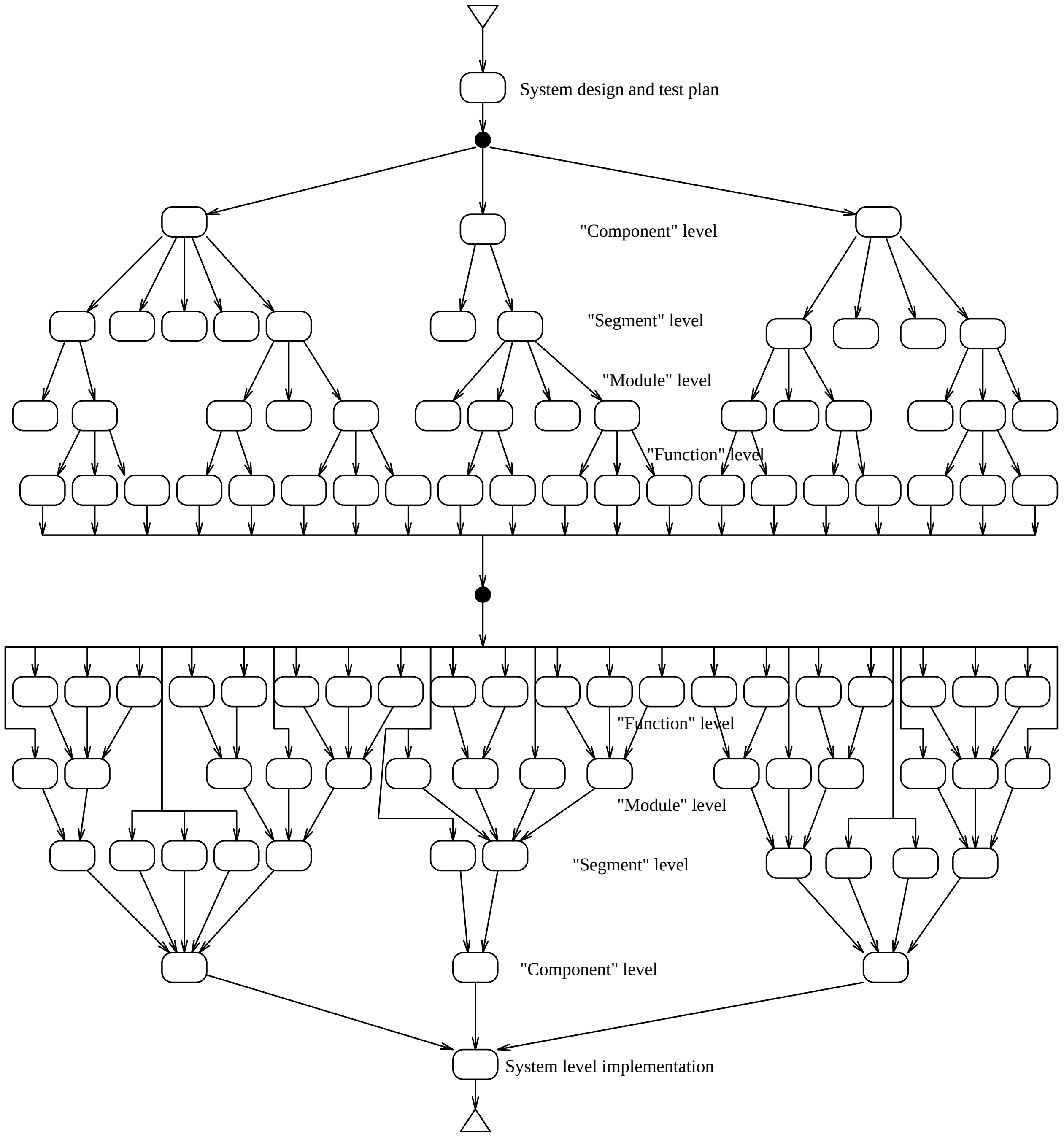

Figure 10: Idealized Design and Implementation

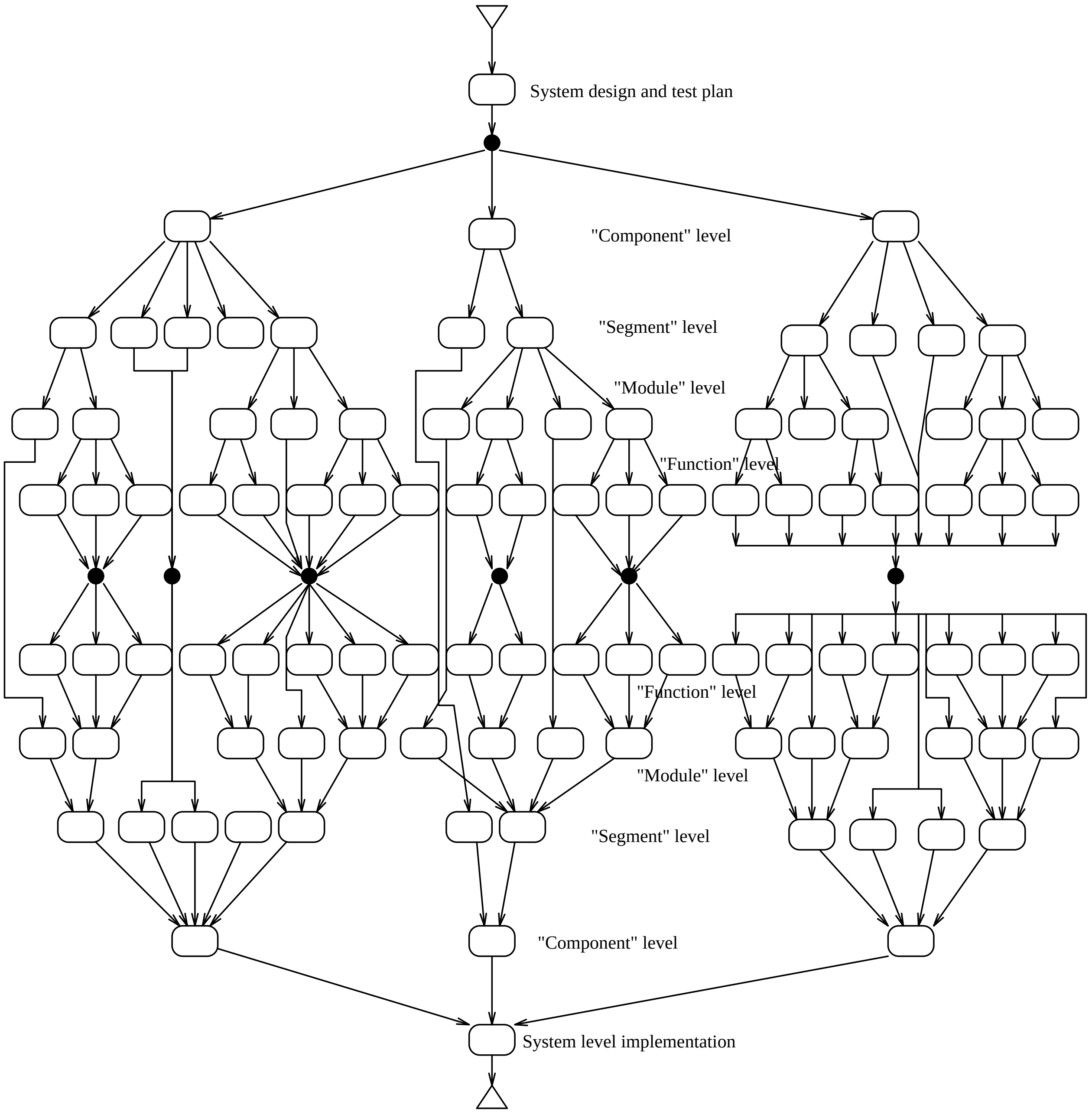

Figure 11: Staged Design and Implementation

the design are reviewed and released for implementation while other parts are still under design. Of course this strategy represents a different sort of gamble; if the design of an apparently unrelated part of the system suddenly becomes related (or even *causes the previously-reviewed and released design to change*, then the implementation work is wasted effort — we discuss this problem below). Thus Figure 11 might represent a more aggressive strategy for staging design and implementation (even though it is inconsistent with the higher level model). Various parts of the implementation are enabled to begin when their respective designs are complete.

This brings out an important point about using models as an aid for thinking and communicating about processes: high level models should be interpreted as an expression of general intent in the same sense that an abstract or executive overview summarizes a report; lower level models may be inconsistent with regard to formal modeling properties while being completely consistent with respect to the intent of the process.

We believe that the purpose of the 2167A specification is to force the bidding contractor to produce software products using a topdown methodology that conforms to the requirements. The contractor is expected to determine a set of requirements from the negotiated statement of work, to produce a test plan that shows how to test each aspect of the requirements *independent of the design*, to create a design that satisfies the requirements and which can be tested, and to implement the design. The problem that bidding contractors have with the process is that it is very difficult to do using a topdown methodology for at least the following reasons:

1. Requirements are rarely complete enough to specify what is really expected from the product.
2. Complex requirements may be inconsistent.
3. Hierarchically-refined designs may uncover design errors at higher layers when lower layers are designed. For example it may not be possible to define a subsystem that meets the higher layered requirements.
4. Design problems may be discovered at implementation time.
5. Changes discovered in detailed design or implementation may impact the work assigned to other organizations; this will cause the other organization to be less efficient (e.g., to go over budget or over schedule), thus such changes will be met with rigorous resistance.
6. Persons responsible for high level contractor resources may not be able to be fully aware of design conflicts.

While these problems can become intractable in large software engineering organizations, it is clear that they are less related to the methodology than to management, organization, and to general intra-contractor politics. The granting organization simply requires that deliverable products be consistent with a negotiated contract. The problem arises due to the lack of understanding of the deliverable by both parties, and from the complexity of the deliverables.

Figure 12 represents a modification of the process that adds feedback links between the implementation and design phases of the waterfall model. When detailed design or implementation uncovers an argument for changing the design, then the appropriate “project manager” actor should determine how the project should handle the problem. The decision may be to ignore the problem by continuing with the original plan; this means that the resulting deliverable may be inefficient,

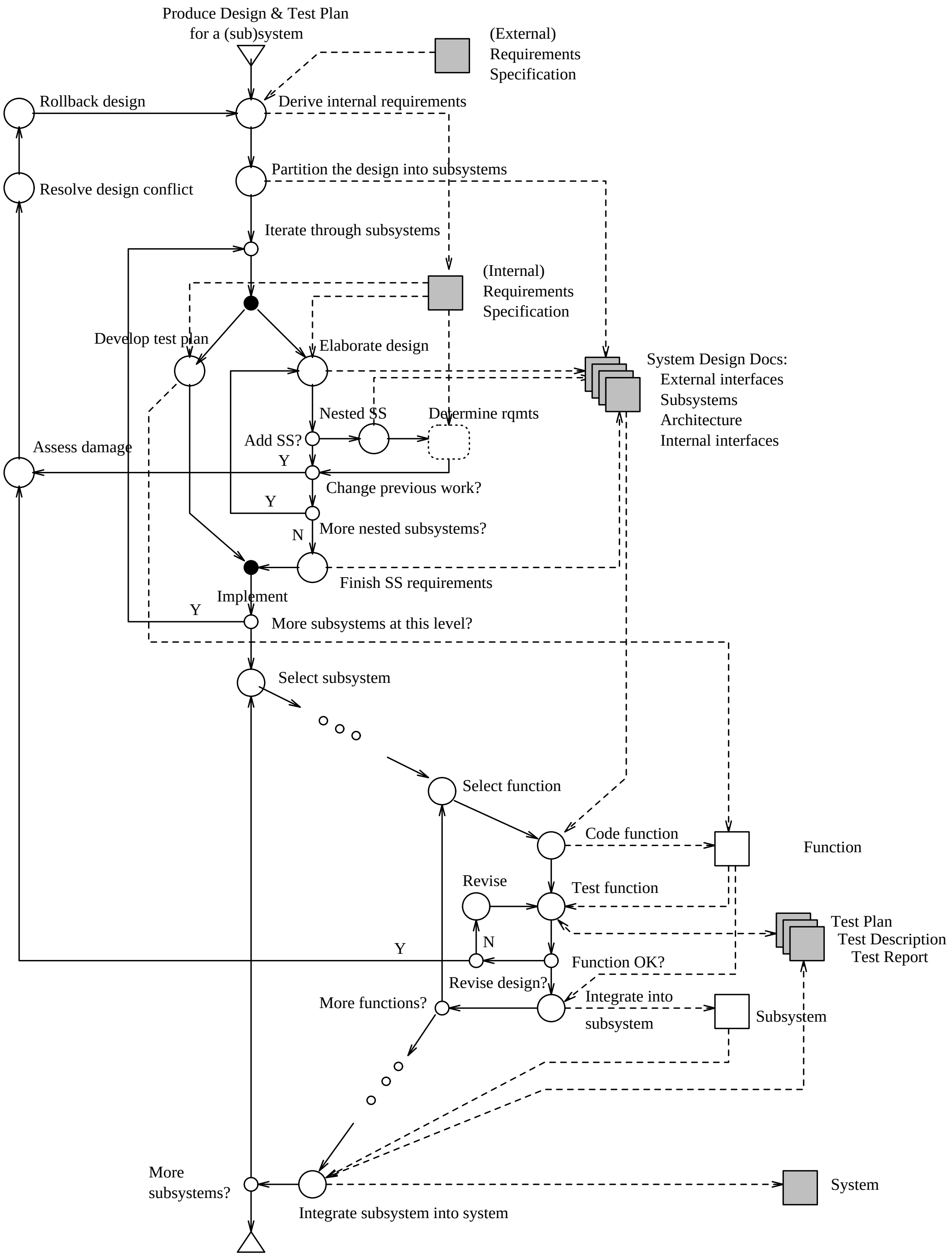
Produce Design & Test Plan for a (sub)system
(External) Requirements Specification
Rollback design
Derive internal requirements
Partition the design into subsystems
Resolve design conflict
Iterate through subsystems
(Internal) Requirements Specification
Develop test plan
Elaborate design
System Design Docs: External interfaces Subsystems Architecture Internal interfaces
Nested SS
Determine rqmts
Add SS?
Assess damage
Y
Change previous work?
Y
More nested subsystems?
N
Finish SS requirements
Implement
Y
More subsystems at this level?
Select subsystem
Select function
Code function
Function
Revise
Test function
Test Plan Test Description Test Report
Y
N
Function OK?
Revise design?
More functions?
Integrate into subsystem
Subsystem
More subsystems?
Integrate subsystem into system
System

Figure 12: Exceptions in Design and Implementation

or may fail some tests. The decision may be to change the design, implying that the impact of the change must be considered (e.g, how many other design units will have to change? What is the current investment in those designs and implementations? etc.) The process must then be rolled back and restarted.

We only show design exceptions arising at two different places — during detailed design and during implementation. They could, of course, arise in other places, with corresponding edges leading to the "Assess damages" activity.

Process models are the messenger, not the problem. Our position is that the more information about the impact of changes that can be presented to decision makers involved in such decision the better. The process model is a language explicitly designed to convey that kind of information to a human.

4 Conclusion

4.1 Using the Model

We have provided a high level process model describing how different parts of an organization can interact to produce a software product along with documentation that conforms with the 2167A requirement. Increasingly detailed models represent increasingly complex interactions among roles — groups — in the organization. The models we have illustrated are insufficient to "automate" the process, and would probably only be of limited use as an automated coordination model to stage work for the engineering organization. However, even at this level of detail they can serve several other important purposes when supported with a system:

Describing the Process. Large software projects frequently have staff turnover — new people arrive as the project progresses into detailed design and implementation, and staff depart after the product is released. In large projects, attrition is also normal, requiring that various staff members be replaced by new people. The model provides a high-level description of how the project is organized. When models of the form used in Figures 6 and 8 are generated, they provide a graphic description of the various levels and units in the design.

Task Assignment. The model is a graphic basis for considering various strategies of organizing work teams and delegating work to them. Analysis tools enable a manager (at any particular level) to experiment with different strategies for assigning work.

Tracking Tool. Auditing tools can also be added to the support system to track the status of the development work. If certain parts of the development begin to lag behind others, then the tracking facilities can be used in conjunction with task assignment facilities to adjust the number of workers focused on any particular part of the project. People responsible for specific units must sometimes make decisions about modifying a unit (recall Figure 12). The model provides a specific artifact to assist that person in making decisions regarding incorporating change, rolling back related development, etc.

Document Management. In a large project, document preparation is very difficult. The full document set must be internally consistent, and support function tracing from requirements to implementation (and the inverse). As designers create documents, they need to be able to navigate through the set of existing and pending documents to produce a document that is consistent with the rest of the system. While it is possible to design a database tool to maintain required relationships among documents, including generating skeletal documents when a design is initiated, it is difficult for a human user to logically absorb all of the relationships at once. The model provides a means by which the designer/implementer can navigate through the existing documents to understand the requirements and/or develop requirements for nested systems that are consistent and preserve traceability.

4.2 Conclusion

This case study has investigated the application of process models (ICN workflow models) to represent a software engineering process. We have not built a system to provide the support described above, although we have built several nontrivial workflow systems for description, modeling, analysis, and enactment. Our previous applications have addressed office applications, business processes, and distributed program organization, while this paper has focused on software processes. Nevertheless, the static model has proven to be useful in discussions with software engineers familiar with development under the 2167A guidelines.

Our research group continues to study process models in an effort to make them more flexible for use in offices [2, 7, 8, 9, 10]. This case study has highlighted the need to be able to add a broad spectrum of different analysis tools to operate on a model, e.g., to manage requirement traces and to produce hierarchically consistent design and test documents. Like our experience with ICN-based enactment systems [4], this application continues to enforce the need for clear and complete mappings of activities to roles and to actors. Again, practical models identify the need for hierarchy in the model, but also point out practical problem with refining the model to include details that interact across subtrees of the hierarchy (e.g., Figure 11 illustrates a set of relationships between elements of the model in Figures 6 and 8 that is not evident in more abstract views of the same part of the process). Finally, the study has emphasized the value of complementary model views for representing parallel activity (we also encountered this in our related work on parallel program models [1]): when one is focusing on understanding the process in general, descriptions analogous to Figure 5 are useful; however, once the model has been derived and it is to be used to study a particular instance of the process, views analogous to Figure 6 are more useful.

Acknowledgement

The author was supported by Grant Number IRI-9307619 from the National Science Foundation. Most of the explanation for how bidding contractors operate under the 2167A guidelines came from Scott Brandt; he also reviewed and critiqued various interim models. The author and Skip Ellis continue to work on refinements to ICNs; that work is reflected in these models and ideas for systems.

References

[1] Adam L. Beguelin and Gary J. Nutt. Visual parallel programming and determinacy: A language specification, an analysis technique, and a programming tool. *Journal of Parallel and Distributed Computing*, 22(2):235–250, August 1994.

[2] Richard Blumenthal. *Representing Unstructured Work Flow Activities by Dynamically Exposing Contextual Information (tentative title).* PhD thesis, University of Colorado, 1994. in preparation.

[3] Grady Booch. *Object-Oriented Analysis and Design with Applications, Second Edition.* Benjamin/Cummings, 1994.

[4] Bull S. A. *Introduction to FlowPATH*, May 1992. Manual No. 44 A2 60XM.

[5] Clarence A. Ellis. Information control nets: A mathematical model of office information flow. In *Proceedings of the 1979 ACM Conference on Simulation, Measurement and Modeling of Computer Systems*, 1979.

[6] Clarence A. Ellis and Gary J. Nutt. Office information systems and computer science. *ACM Computing Surveys*, 12(1):27–60, March 1980.

[7] Clarence A. Ellis and Gary J. Nutt. The modeling and analysis of coordination systems. In *ACM 1992 Conference on Computer-Supported Cooperative Work*, 1992. workshop position paper.

[8] Clarence A. Ellis and Gary J. Nutt. Modeling and enactment of workflow systems. In *Advances in Petri Nets 93*, pages 1–16, June 1993. Invited paper.

[9] Clarence A. Ellis and Jacques Wainer. A conceptual model of groupware. In *ACM Conference on Computer Supported Cooperative Work*, 1994. To appear.

[10] Clarence A. Ellis and Jacques Wainer. Goal based models of collaboration. *Collaborative Computing*, 1:61–86, 1994.

[11] Heinrich J. Genrich. Predicate/transition nets. pages 3–43, 1986.

[12] *Proceedings of the IEEE Software Process Workshop.* IEEE Computer Society Press, 1993.

[13] IEEE Software theme on Mature Processes, July 1993. Robert Lai, Guest Editor.

[14] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use, Volume 1.* Springer-Verlag, 1992.

[15] Jintae Lee, Gregg Yost, and the PIF Working Group. The PIF process interchange format and framework. Technical report, University of Hawaii Information and Computer Science Department, November 1994.

[16] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[17] Gary J. Nutt, A. Beguelin, I. Demeure, S. Elliott, J. McWhirter, and B. Sanders. Olympus: An Interactive Simulation System. In *1989 Winter Simulation Conference*, pages 601–611, Washington, D.C., December 1989.

[18] Gary J. Nutt and Paul A. Ricci. Quinault: An office environment simulator. *IEEE Computer*, 14(5):41–57, MAY 1981.

[19] Joseph F. Traub, editor. *The National Challenge in Computer Science and Technology.* National Research Council, National Academy Press, 1988.

[20] U. S. Department of Defense. *Military Standard, Defense System Software Development*, 1988. DOD-STD-2167A.

[21] A workflow management coalition specification: Glossary. Technical Report Document Number TC00-0011, Workflow Management Coalition, Brussells, Belgium, August 1994.