Proceedings of ICAD2000
First International Conference on Axiomatic Design
Cambridge, MA – June 21-23, 2000
ICAD062

INTEGRATION OF AXIOMATIC DESIGN AND PROJECT PLANNING

Duane Steward, DVM, MSIE, PhD
duane@mit.edu
Clinical Decision Making Group, Laboratory for
Computer Science, Massachusetts Institute of
Technology, Cambridge, Massachusetts

Derrick Tate, PhD
dtate@axiod.com
Axiomatic Design Software, Inc.
221 N. Beacon St.
Boston, MA 02135 USA
+1 (617) 746-9267

ABSTRACT

Software development projects require the translation of good abstract ideas into clear design specifications. Subsequent delivery of the software product in moderate-to-large scale projects requires effective project planning and assignments for a team of software engineers to meet deadlines in the presence of resource constraints. This paper explores the hypothesis that axiomatic design may be integrated into the process of project planning and task assignment for software development teams. An approach to mapping functional requirements and design parameters into tasks of a project plan Gantt chart is described. Effects of transferring the relationships of design matrices to task links are discussed. The result is considered by the authors to be a productive integration facilitating the rapid delivery of product by software engineering teams.

Keywords: Axiomatic Design, Project Planning, Software Development, Gantt Charts, Resource Management.

1 INTRODUCTION

Axiomatic design provides a systematic approach for generating detailed specifications for software product design.[1,2] Using this approach, details are derived from what often is originally nothing more than abstract ideas in the minds of creative individuals. As a consequence of applied axiomatic design, functional requirements (FRs), design parameters (DPs) and design matrices representing the design interactions between elements are articulated. Knowing such details is an essential step in the definition of software products, but it is a prescriptive step. Software developers interested in delivering the product must still implement the design successfully.

Simple projects may be implemented without well-defined organization. However, software development involving numerous programmers, quality assurance protocols, rigorous documentation, collaboration with supplemental technologies, consultants and support personnel, requires organization of collaborative efforts and resource allocations. Optimizing the assignments of project sub-components and workflow management can make significant difference in the delivery time of software design. In the competition to be first to market, the design and execution of a development plan can be the difference between success and failure to deliver in a timely fashion.

Implementation of software design benefits from decomposition much the same as the design process itself. The process of breaking up the implementation of a design into smaller steps (i.e., "tasks" in project planning) is as essential to executing a design as it is to the design's creation.

Not all software engineers are alike. Programmers differ in experience, speed, breadth of knowledge and composition of skill sets. Management of human resources is essential to optimized product delivery times. Identifying the tasks necessary to fulfill a design and matching the best available human resources to those tasks is the responsibility of the software project manager. Furthermore, time estimates for product delivery and intermediate milestones can be accomplished using such task lists. The reality of software engineering includes the requirement to identify tasks and make appropriate assignments for a team of diverse software engineers.

Time constraints for software development are growing in importance. Optimized software development life cycles require attention to project planning. Interdependencies between tasks of a project plan often create rate-limiting sequences in the overall plan. The least possible time to delivery is must be known for purposes of marketing, financing and barriers to competition in addition to human resource management.

Currently, project planning in software development involves the identification of modules in the software architecture that cluster related elements in the software design. The same modularity that promises interchangeable parts and component reuse serves as a natural partitioning for distributing the tasks essential to the product's development. Tasks are assigned in a way that makes sense of the fit for at least most of the engineering team members.

Typically project planning involves some degree of identification of dependencies between tasks identified in the project plan. However, without a systematic approach to the explication of dependencies, omitted links are likely. Most project planning understates the dependencies inherently present in a project.

It is the author's suspicion that this could have a lot to do with why so many project plans fall apart as the project ensues. As the project approaches its completion, the timing and order of tasks gets further and further from the plan's representation and often results in plan abandonment.

Nonetheless, time estimates for intermediate milestones provide a mechanism for monitoring progress. Of course, knowing when to expect completed milestones can enable adaptive measures midstream in the development process, e.g., additional software engineers may be hired if numerous intermediate deadlines are missed. A good project plan, consisting of clearly articulated tasks, assigned to specific development team members with time to completion estimates is thus key to project planning and adaptive execution of design implementation.

1.1 HYPOTHESIS

The functional requirement hierarchy of the axiomatic design paradigm may be viewed as a precursor to a functional specification in software development. The hierarchy is inherently an outline of functionality consisting of short descriptors. If each of these descriptors is explained in a short paragraph, a functional specification results. This result will contain all necessary and sufficient details of the design to the same degree that the functional requirements contain the necessary and sufficient list of requirements.

Similarly, the design parameters of an axiomatic design approach map into a design specification document. Each briefly described parameter of design can be expanded into a description of the signature of the software module, package, class object, method or property represented in the design parameter. Coupled with a description of justification for interactions asserted in the design matrix, the result should be adequate to hand off to programming staff or contractors. To the degree that the design parameters map to the functional requirements, the code delivered by programmers should fulfill the expectations represented in the functional specification.

Applied axiomatic design, then, should be able to produce specification documents that fit the classic paradigm of Quality Assurance allowing QA staff to evaluate the product against the functional specifications, or more fundamentally, against the articulated user needs or functional requirements list.

This hypothesis is being put to the test in a production environment of an Internet startup company. The company's goals involved an object-oriented programming approach to the design and implementation of a novel product. An ambitious goal for release of a multi-tier software product in six months was adopted. This was made possible in part by the details rapidly generated by applying axiomatic design. At this writing, prototype software has been completed to validate assumptions and expose deficiencies. Functional and design specification document writing will ensue. If functional specification and design specification documents suitable for QA can be constructed from these elements of axiomatic design, the above hypothesis will be supported. If an operational project plan can be derived, these assertions will be further strengthened. Such success will at least establish the feasibility for this approach to design. Sharing the early experience in application of this design paradigm in project planning and thus testing the hypothesis is the intent of this paper.

At this time, a project plan constructed from the DPs is in place guiding the work-in-progress. The project plan is expected to be more viable as a result of the extensive linking between tasks captured by the inclusion of design matrix relationships. The level of detail in functional requirements early in product development and the abundance of links in the decoupled design have had interesting effects on the overall project. The level of detail has surprised collaborators and consultants brought into the project. The abundance of links has made resource allocation more difficult to level and squeeze into minimal time frames. However, the plan is expected to persist where other plans become obsolete before the project is complete.

2 METHODS

The first author derived FRs from the company's business plan and interviews with domain experts. Matching DPs and Design Matrices were constructed in adherence with the roadmap described by Tate[3],[4]. Particular attention was paid to deriving physical design parameters that, for each FR, answered the question, "By what means will this Functional Requirement be fulfilled?" For instance, "a web page with pull down selection list" was asserted as the design parameter for "provide a means for the user to select one of multiple [options]." Design matrices were constructed with particular attention to design interactions, e.g., a change in the *ith* DP does or does not have an impact on the *kth* FR. The resulting DPs were directly input into the Gantt chart of a project plan as individual tasks. The interactions present in the design matrices were then represented as links between tasks in the Gantt chart.

By adding time estimates to the individual tasks and making assumptions about the human resources (i.e., size of programming staff), the Gantt chart takes on a common appearance of tasks distributed over time with internal dependencies.

The nature of the hierarchical "zig-zag" derivation of FRs and DPs is such that multiple DPs may be physically integrated into a single component. The meaning in this relationship is that the physical component described in the DP is being used to fulfill more than one FR. However, it would be erroneous to list a task creating that component more than once in the project plan. Therefore, the task list in the project plan must be consolidated to remove redundant listings.

If the FR/DP decomposition is carried out to a significant degree, resource assignments may be rather granular in size and scope. Planning may benefit from clustering tasks so that one person can work on a number of related tasks to improve coherence in design and implementation. So, DPs may be grouped into related families. Members of these groups may have explicit ties in the design matrix or heuristic ties not explicitly represented. Relationships that do not involve dependencies (e.g., shared technology or resource requirements) may be the basis for such clustering.

One very relevant example of relatedness useful for clustering is the case where two or more components involve the same programming specialty, e.g., XML programming or dynamic HTML. They are related by common technology, but not dependent upon one another. In the application layer of a multi-tier Internet architecture, some components downstream may be on parallel paths of a dependency tree making it possible to assign them to different programmers after the common prerequisite is completed. However, the relatedness of the components may beg for assignment to the same programmer under a rationale of continuity.

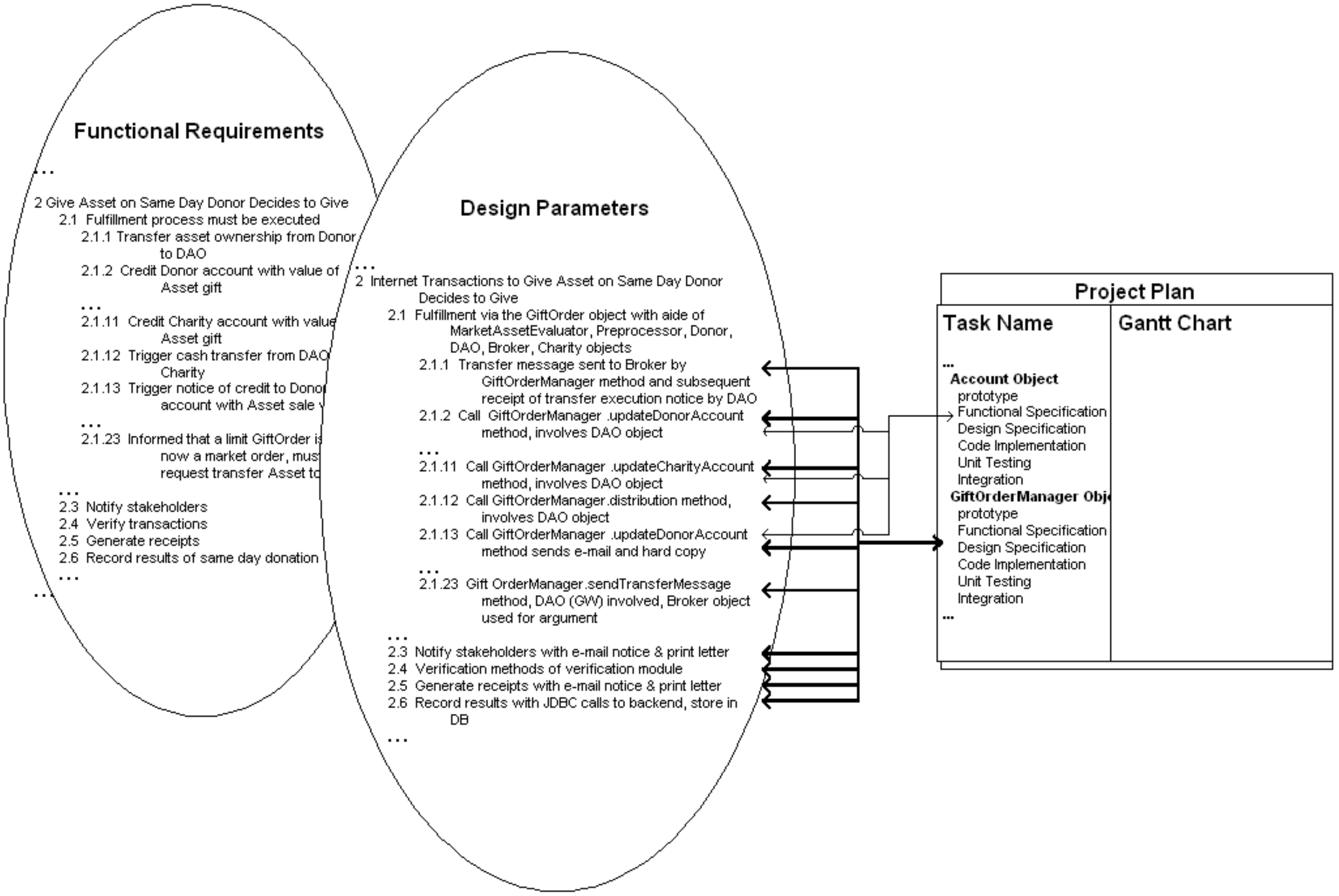
Functional Requirements
...
2 Give Asset on Same Day Donor Decides to Give
2.1 Fulfillment process must be executed
2.1.1 Transfer asset ownership from Donor to DAO
2.1.2 Credit Donor account with value of Asset gift
...
2.1.11 Credit Charity account with value Asset gift
2.1.12 Trigger cash transfer from DAO Charity
2.1.13 Trigger notice of credit to Donor account with Asset sale v
...
2.1.23 Informed that a limit GiftOrder is now a market order, mus request transfer Asset to
...
2.3 Notify stakeholders
2.4 Verify transactions
2.5 Generate receipts
2.6 Record results of same day donation
...
...
Design Parameters
...
2 Internet Transactions to Give Asset on Same Day Donor Decides to Give
2.1 Fulfillment via the GiftOrder object with aide of MarketAssetEvaluator, Preprocessor, Donor, DAO, Broker, Charity objects
2.1.1 Transfer message sent to Broker by GiftOrderManager method and subsequent receipt of transfer execution notice by DAO
2.1.2 Call GiftOrderManager .updateDonorAccount method, involves DAO object
...
2.1.11 Call GiftOrderManager .updateCharityAccount method, involves DAO object
2.1.12 Call GiftOrderManager.distribution method, involves DAO object
2.1.13 Call GiftOrderManager .updateDonorAccount method sends e-mail and hard copy
...
2.1.23 Gift OrderManager.sendTransferMessage method, DAO (GW) involved, Broker object used for argument
...
2.3 Notify stakeholders with e-mail notice & print letter
2.4 Verification methods of verification module
2.5 Generate receipts with e-mail notice & print letter
2.6 Record results with JDBC calls to backend, store in DB
...
Project Plan
Task Name
Gantt Chart
...
Account Object
prototype
Functional Specification
Design Specification
Code Implementation
Unit Testing
Integration
GiftOrderManager Obj
prototype
Functional Specification
Design Specification
Code Implementation
Unit Testing
Integration
...

Figure 1. An example of the application of *Axiomatic Design* to project planning. The design is for an internet based product that enables the donation of stocks to charities. In *Axiomatic Design*, the *Functional Requirement (FR)* heirarchy maps to the *Design Parameter (DP)* heirarchy. DPs map to tasks of a project plan in turn. Multiple FR/DP pairs may relate to a single component of software listed as an individual task of the project plan. Dependencies captured in the *Design Matrix* of *Axiomatic Design* (not shown) are used to justify links in the Gantt chart of the project plan.

The outcome of consolidation and clustering is a list of tasks for the project plan with links. The one-to-one mapping of FRs to DPs, however, is missing.

3 RESULTS

Ninety-seven FR/DP pairs in a tree with 5 primary branches at the root of each hierarchy were transformed into tasks of a project plan. All the DPs were first pasted as tasks into a project planning software$^{\alpha}$ without alteration. All interactions captured by the Design Matrix analysis were represented as links between tasks. The list of tasks was reviewed for redundancy and duplicates deleted being careful to preserve all dependency links in the single remaining task representation. Tasks were thereafter clustered into groups according to similar topic domains and skills required for coding.

The result was an inadequate representation of single unit software development because it lacked a breakdown of each unit's development into functional/design specification, code implementation, unit testing and quality assurance (QA) evaluation. This was easily remedied by decomposing the original single tasks into sets of subtasks properly representing these stages of unit development.

An additional expansion of tasks into subtasks occurred where junior programmers needed the single task broken down into a migration path from simple to sophisticated fulfillment. These localized migration paths were not required to fulfill the FR/DP pair decomposition or Design Matrix analysis. However, they are a useful strategy for enabling senior programming staff to clarify expectations and monitor progress of junior staff.

The outcome of consolidation, clustering and expansion was a list of 275 tasks organized into 40 groups. Time estimates were established for each task. This enabled the computation of the estimated total person-hours required for the project. At this point, the model was used for rough estimates in strategic planning for selecting a release date and staffing estimates for gross budget projections.

The granularity of the tasks ranged from simple properties of software objects to loosely defined modules. Only components that represented future development pathways were allowed to remain loosely defined. Most DPs were in terms of specific web

$^{\alpha}$ Microsoft Project 2000

pages, specific application layer objects in the development object model, encapsulated methods within specified objects or database table schemas and sub-schemas. Properties of objects (e.g., a Boolean field to serve as a status flag) were rolled into task clusters with larger related entities. Because they represent such a fine granularity, they were not given independent development cycles (i.e., specification documents/code implementation/QA).

A Gantt chart was generated using the project planning software. Human resources were assigned to the tasks and resource leveling applied (distributing the tasks of the Gantt chart over time under a constraint that no resource may be used more than 100% in any day). Resource leveling distributes assignments of the same resource over time giving priority in accordance with dependency links and manually assigned scalar values. Design Matrix interactions represented as links between tasks nearly eliminate the need to assign scalar values.

Specific staff were assigned to tasks in the project plan. Overall project time estimates were made. Time estimates for individual tasks on the critical path sum to the shortest possible time to project completion. If this interval was not acceptable, additional staff were added (at least on paper for planning purposes). The assignments made to individual programmers were compared to evaluate and evenly distribute the workload according to estimated time assumptions. At this point, software elements that might otherwise be developed independently became bound by sharing the same human resource. Revisions were made to overall project completion estimates based on these realities of specific staff assignments.

Milestones were sought for the purposes of reporting progress to other company departments. Categorical analyses of FRs were the most intuitive way and thus preferred way to choose a logical sequence of milestones. FRs of highest priority in the eyes of executive officers and marketing strategists could easily be identified. However, the project plan no longer was a direct mapping of the FRs since the project plan was structured by components. Identification of milestones in the project plan marked by the completion of tasks were not a clear and direct representation of priorities in terms of FRs. With some auditing of the transformation of original DPs into project plan tasks a coherent sequence of milestones was identified (better done as part of the process than after the fact).

Identified milestones were grouped according to some approximation of the priority expressed with FRs. The first of these were earmarked as a first prototype. This was followed by a sequence of additional milestones representing groups of additional features. Dates for expected milestone arrivals were easily extracted from the plan based upon the completion of the last task in each group.

It was anticipated that the milestones would naturally fall into proper order as a result of this methodology. However, it turned out to be difficult to arrange the project plan in such a way that these tasks were accomplished in an order consistent with their priority. The reason for this will be discussed shortly.

4 DISCUSSION

The resulting milestones were grouped into a bundle of essential features and regarded as a fundamental prototype that would serve to inform the process of writing functional and design specifications. Subsequent development was thereby guided by both prototype experience and documented expectations. Projections, budgeting, staff recruiting and assignments were dealt with in terms of these experiences and expectations. Perhaps the greatest advantage of identified milestones was the ability to focus and identify events that would mark satisfactory progress of the project in terms that were closely related to the FRs identified earlier in the development process.

The most remarkable features of the application of axiomatic design to the construction of project plans are the early delivery of detail in identified tasks and the extent of interactions captured as links between tasks. As the startup company wrestled with decisions between outsourcing and in-house development, sharing design progress to-date with potential collaborators, they frequently received feedback that consultants were not accustomed to so much detail at such an early stage of development. The level of detail in the FR/DP hierarchies made it easy to lay out an equally detailed project plan. The links between tasks were numerous.

With so many links between tasks in the project plan, resource leveling and minimization of overall project completion time becomes an interesting challenge. This is especially true if the milestones identified as they were here are to be achieved in the order of priority. The dependencies between tasks tend to increase the project completion time. These dependencies are a result of both design matrix analysis and coherent human resource assignments where the staff size is small with a few key people. One general approach to reducing project time is to assign more personnel. Additional personnel may be assigned tasks that either have no dependencies or depend only on tasks that can be completed early in the project plan. These tasks tend to involve functionality that spin-off the core ambitions of the project. Although these activities are easy to distribute among new staff additions, they do not help accomplish the more essential features before the less essential features. Hence, the additional staff only enable an earlier completion of the less essential tasks. A higher degree of relatedness between tasks results in a project plan with milestones that may not fall into the same order over time as that derived from perspectives uninformed by the design matrix of interactions. No amount of additional staff will reconcile the order of milestone achievement so long as the dependencies exist (in a decoupled design).

It is the authors' suspicion that this approach to project planning results in dependency integration earlier than in than less systematic alternatives for identification of task links. Perhaps this offers an explanation why so many project plans fail to remain viable once the project gets well under way. The dependencies are discovered along the way in those cases and the need to revise the project plan becomes overwhelming in the face of deadlines for delivery. Where the dependencies are accurately captured and factored into project planning this early in the project development cycle, we expect the plan to be more robust in the face of realization.

It should be noted that the tasks of a project plan are derived from the DPs and not the FRs of the axiomatic design paradigm. In this approach, design parameters map to the tasks of the project plan. While FRs are not very definitive for a project plan, DPs are. Engineers look to the project plan for what they need to accomplish next, to monitor the expected progress over time or ascertain dependencies. Descriptions of functionality do not serve this purpose, but the assertions regarding how those functionalities will be fulfilled do. That is precisely what a design parameter is in axiomatic design. In this perspective a hierarchy of design parameters might be loaded into a project plan as tasks along with the dependencies as an early model for project completion, resource requirements enumeration and analysis. Two grids are suggested. One grid would be the FR/DP pairs mapped to skill sets required. Another grid possible is one mapping the FR/DP pairs to personnel already in the company to reveal how much of the project could be accomplished with in-house skill sets or otherwise require outside contractors.

When developing a novel product with no precursor to inform the design process, it is difficult to develop without prototyping. Prototyping helps to define issues and expose oversight as well as prove the concept. DP's, and to some degree, FR's may be altered by the prototype results. It is not likely that the intuition prior to prototype will describe accurately the final details of the product. Following the prototype construction, FRs and DPs may need to be reviewed and altered in light of the things learned.

A final limitation in this approach to project planning is that the plan is only as good as the assumptions made in the axiomatic design process. Omissions and erroneous assumptions made in regard to functional requirements and interactions are not exposed or compensated for by project planning derived therefrom.

Before leaving the discussion it is worth pointing out that DPs could arguably be implemented in an order that differed from the partial order dictated by the dependencies embodied in design matrices. To the degree that the design principles are adhered to, the design matrix will represent a partial order for design of components. Thereafter, any order of implementation that conforms to the design should work without many interactions between design steps, but which have no interaction outside of design, i.e., during implementation. While this is true, managing an engineering staff with two or more project plans (one for design including functional or design specification writing) was not done. It was decided that for this project, it would be an overwhelming amount of detail to maintain and its significance minimal.

It is also acknowledged that this approach to project planning has shown merit in a specific case of rapid software development. Other domains may produce alternate results.

5 CONCLUSION

In this project it has been shown that design parameters of Axiomatic Design may in fact be used to originate a project plan. This plan is being used actively in software development. The effects of doing so include an early identification of dependencies that result in a plan that does not respond to the addition of staff as expected. Further evaluation is in order to substantiate the merits in extended domains.

6 ACKNOWLEDGMENTS

This project has been supported by AssetStream Corporation in the course of their initial software product development as an Internet provider of charitable giving services.

7 REFERENCES

[1] S.-J. Kim, N.P. Suh, and S.-G. Kim, "Design of Software Systems Based on Axiomatic Design," Robotics & Computer-Integrated Manufacturing, Vol. 8, No. 4, 1991, pp. 243-255.

[2] Suh, N.P. (1990) "The Principles of Design." Oxford University Press, New York.

[3] Tate D., Nordlund M., "A Design Process Roadmap as a General Tool for Structuring and Supporting Design Activities", Proceedings of the Second World Conference on Integrated Design and Process Technology (IDPT-Vol. 3), Society for Design and Process Science, Austin, TX, pp. 97-104, Dec. 1-4, 1996.

[4] Tate, D. (1999) "A Roadmap for Decomposition: Activities, Theories, and Tools for System Design." Ph.D. Thesis, Dept. of Engineering, Massachusetts Institute of Technology, Cambridge, MA USA. February 1999.