2.0 Basic Software Product Engineering

Software product engineering is the performance of a project to provide a designated simple market with a product that addresses the perceived needs of customers in that market. Such needs are expressed in terms of capabilities that the product will enable for the operation of a customer enterprise.

The context in which a project operates reflects the broader concerns of both provider and customer enterprises. In general, the targeted market will be a set of customers that have similar enough needs that a single product can be built that will meet the needs of any of them. In particular cases, the market for a product may consist of only a single customer. Although typically the development project is part of a separate provider enterprise, it may in the case of a single customer market be an organization within the customer enterprise.

A project is initiated as an agent of an enterprise-chartered program, assigned to work with one or more designated customers that program marketing has identified as having needs for a product within the scope of the program's charter and capabilities. A project is envisioned as a mutually beneficial collaborative relationship with customers, resulting in the realization of a product responsive to each customer's current and future needs.

A product is never a static construct: the needs it is meant to satisfy may be poorly understood initially and will change over the expected useful life of the product. A project must build a product in such a way that likely changes, whether due to improved understanding of actual needs or subsequent changes in those needs, will be aspects of the product that are the easiest to change. Further, as changes are made over the product's useful life, the ability to make likely future changes needs to be sustained.

The way a product is built is defined by the development process that a project follows. A development process is defined in terms of a set of activities, methods and practices associated with each activity, and information dependencies among activities.

Traditional development approaches specify that activities should be performed in a more or less fixed order. Conversely, this chapter suggests a concurrent process for software product engineering. A concurrent process, which allows work on any activity to proceed whenever needed information and resources are available, is how traditional approaches are actually made to work in practice.

A project is defined by a "project model" consisting of a "project management model" and a "product model". The project management model defines how the project operates to create the product model that defines the product. The project model has two purposes: prescriptive (specifying the operation of the project and the composition of the product) and predictive (providing a basis for estimating the properties of the project and product).

The remainder of this section defines the elements of the project model, a concurrent engineering process based on that model, and a process quality framework for characterizing project productivity. The section following this one describes the context in which a software development project operates, while the remaining sections of this chapter describe the elements of each facet of the project model.

The Software Project Model

The ***project model*** for a software product engineering project has two elements: a project management model and a product model (Figure 2.3-1). The premise for a project model is the competence and conventions imparted to the project by its chartering program and the capabilities needed by its designated customer. The project management model consists of a single facet whereas the product model is partitioned into seven facets (i.e., topical models). Each facet of the product model describes the product from a particular perspective, based on the developer competence needed to define its content.

Each facet of the project model consists of a set of elements that partitions the information content of that facet. An activity is associated with each element for determining its content, according to the nature (purpose, subject matter, and

dependencies) of that element, tailored as appropriate to the project's scope. The resulting set of activities specifies the software development process for the project.

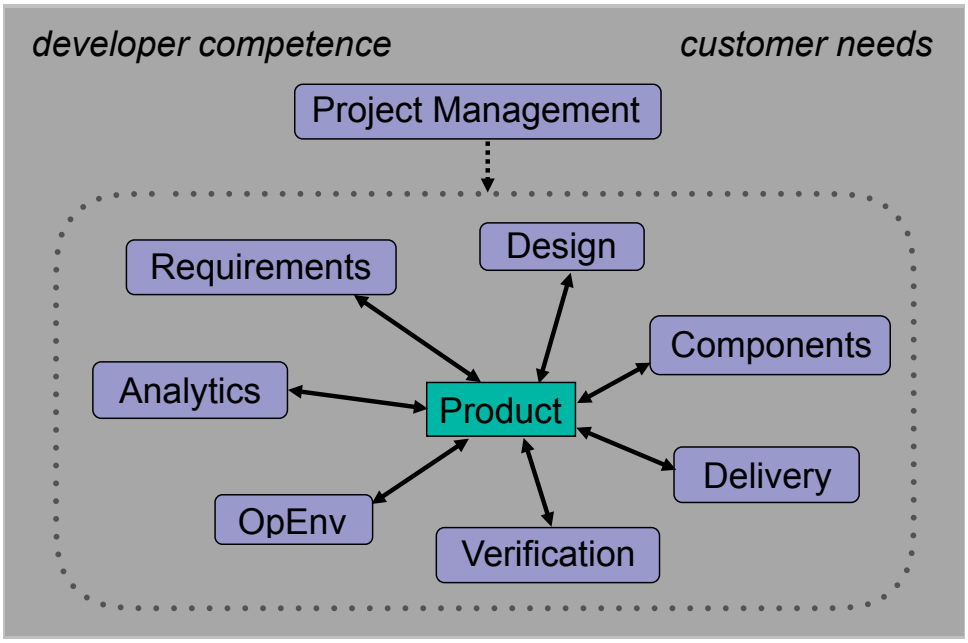

Figure 2.3-1. The Faceted Structure of the Software Project Model

The ***project management model*** describes the organization, resources, and performance of the process by which a product is created. The project management model specifies the structure of the product model and the product engineering process based on that structure. Tasks are initiated and performed in accordance with the criteria defined by the activities associated with each of the various facet-associated elements of the project model. Each activity is specified as guidance for corresponding tasks in terms of the objective, needed competence, and any associated method or practices to follow, but differing in the specific goals and subject matter scope on which a given task is focused.

The ***product model*** in aggregate is a designated set of materials (documents, specifications, and models), both developmentally-useful and operationally-useful, by which all aspects of the product are described and realized. The product model holistically expresses all information about a product, consistent with the nature of the product as the subject of development and subsequent use by a customer. Many elements contain information that constitutes a "specification" for the content of other elements, each such element being a "realization" of that specification. Consistency of a

product model requires that the elements in each specification-realization relationship have consistent content.

The content of each (topical) model facet is defined strictly, in terms of the topical-specific questions about the product that it is expected to answer and its intended audience. For example, customer needs are expressed in terms of the capabilities that the customer needs a product to support; conversely, product requirements specifies the developer's view of the behavior that the product is expected to exhibit. These are complementary views of essentially the same information and as such must be consistent for a product to be considered complete.

The content of the topical models and their elements are conceived to coherently partition product information, by which both inadvertent omission and duplication of (equivalent or potentially conflicting) information can be reduced. Additional supporting information is associated with each model element to specify the basis for the derivation of its specific content, including assumptions made, alternatives considered, rationale for choices made, and potential changes that might be needed in the future and how these would affect the product.

Project management defines tasks by assigning resources to develop content associated with one or a related set of activities for a specified scope of information. For example, a task may be initiated to perform the activity of describing particular capabilities that the customer needs the product to support. However, a task may also involve performing multiple related activities, not limited to the content of a single project model element but possibly involving development of content associated with related elements of the same or different facets. For example, a task to describe a particular product capability in the customer needs element of the delivery model may also include defining those portions of requirements model elements that specify the associated observable behavior of the product that realizes that capability. The order in which such related activities of an assigned task are performed is not prescribed but is left to the developer's judgment (e.g., requirements model content may be defined before or after corresponding customer needs content, as long as the end results are mutually consistent).

Any dependency between two elements must be resolved in a way that reconciles the goals of both. When the content of an element changes, it can trigger changes in related elements in order to maintain consistency. Most significantly, the development of a realization element may expose flaws or produce conflicts with a related specification; the resulting resolution between them can lead to a cascade of changes through other elements. (For example, the requirements model is a specification for the design model; if it specifies an infeasible design, the specification would need to be revised).

The following briefly describes each of the facets of the project model, according to the nature of its content, its objective, and needed developer competence, Subsequent sections will describe the elements comprising each of these facets in more detail including (e.g., specification-realization) dependencies among related elements.

- **Project Management**: A specification for how the development team is resourced, organized, directed, and coordinated for the purpose of building an envisioned product.
  (*objective*: specify a plan with allocation of project resources to tasks, consistent with project goals and the dependencies among the product perspective models)
  {*competence*: how to orchestrate a project, pursuant to program objectives, to build, deliver, and sustain a software product that meets the evolving needs of a designated customer (or simple market)}

- **Product Requirements**: A specification of the externally observable behavior that the product is prescribed to exhibit within its environment, consistent with customer-envisioned capabilities and constraints
  {*objective*: specify product behavior that is sufficient to satisfy customer-perceived needs}
  {*competence*: how to conceive and specify the behavior of a product of suitable quality that will provide capabilities that support the objectives of a customer enterprise}

- **Product Environment**: A specification of the environment as an information space in which the product will operate and the external entities (users, devices, and systems) that are affiliated producers and consumers of information;

behaviors of external entities that induce effects on the environment; and the behavior of the ecosystem as a whole (the environment and contained entities) as the medium that influences software product behavior.
{*objective*: specify the relevant elements and behavior of the operational environment (product context) within which the business process/procedures/practices/capabilities are applied to pursue business objectives, including scenarios that represent ecosystem behavior}
{*competence*: how an environment and associated entities behave as an information space}

- **Product Design**: A specification of the internal organization of a product and its interactions with its environment and other entities appropriate to realize requirements-specified behavior
  {*objective*: specify the composition of a product that satisfies a best/approximate satisfaction of behavioral quality criteria and provide the means to compose its realization from needed components}
  {*competence*: how to envision the composition of a product that can be built to provide specified behavior and the means by which such a product is realized given appropriate components}
- **Product Analytics**: A specification of the quantitative and qualitative analyses used to determine whether a product will meet behavioral quality expectations and to understand how problem-solution alternatives differ
  {*objective*: specify the nature and tradeoffs of quality criteria, the effects/implications of these, projected emergent behaviors, and how changes would affect these}
  {*competence*: how to characterize quality factors that a product should exhibit and how to evaluate its degree of success in that based on alternative product models}
- **Product Components**: A specification of components (fabricated, built, or otherwise obtained) with which a product can be built
  {*objective*: specify the design, implementation, and verification of each component

needed to realize a product}
{*competence*: how to build or acquire a software component needed to realize a design-specified product, including competence in the subject matter of a component}

- **Product Verification**: A specification of criteria and resulting evidence for the degree to which a (partial or complete) set of product model elements is consistent and complete in meeting prescribed product quality criteria, based on (1) element-associated peer and expert reviews, (2) testing based on scenarios that mimic operational interactions, and (3) application of formal analytic methods
  {*objective*: specify how product model facets are determined to be consistent and complete}
  {*competence*: the means by which a product, or constituent parts thereof, can be empirically evaluated as consistent and complete relative to specified criteria}
- **Product Delivery**: A specification of a verified operationally-useful version of the product, including associated specifications of customer needs, actions for transitioning customer enterprise operational practices, validation and certification criteria and results, actions for product deployment into operational use, and continuing feedback to the project regarding issues in use and potential improvements.
  {*objective*: specify how to explain and deploy a product into a customer enterprise in a form that users can employ to the benefit of their operations}
  {*competence*: the ways in which customer enterprises operate at all relevant levels, the mechanisms by which a product is deployed into operational use, and the means by which the organizational and individual practices of such operations can be transitioned to enable effective use of a product's capabilities}

A Systematic Process for Product Engineering

The purpose of a well-defined process is predictability of effort required to produce an acceptable product. An engineering process defines activities that can be performed to develop, deploy, and sustain a product. An activity is understood in terms of a specified method or practices to be followed: actions to be taken, information needed to do so, and information produced as a result. Specific methods are not prescribed by the process envisioned here, only framed in terms of the essentials of good practice that any appropriate method, as variously defined elsewhere in the software engineering literature, will satisfy. In this process, an appropriate activity is associated with each of the several elements of each facet of the project model, its purpose being to determine the content of that element.

A project is the coordinated performance of a collection of tasks (i.e., work assignments), each task being the undertaking of one or more related activities with an associated allocation of scope and resources. The result of performing tasks in accordance with the engineering process is the aggregate of information produced (i.e., the project model).

For an initial simplified understanding of how work might flow through various minimally essential elements of the project model, the following is a simple linear conception of a work flow through those elements. A realistic work flow is more complexly nuanced with tasks that are more granular and concurrently interwoven. In addition, because the described process permits even related tasks to be performed concurrently, a realistic work flow need not be linear in performance. Each activity may have many tasks associated with it, such as developing various components, evaluating various specifications as work progresses on them, or testing scenarios for different aspects of product behavior. Similarly, there may be repeated iterations of a task during the course of the work flow (e.g., requirements will be iteratively refined, resulting in further work on other elements of the product model). Although deploying a product into customer operational use must be the final task for a given increment with all other tasks having been completed, it may have been preceded from a developer perspective by the initiation of another overlapping increment of development already in progress when the deployment task occurs, such that tasks of a subsequent increment continue.

1. [Project management] a project is instituted to define a product master plan
2. [Project management] a product increment is specified with tasks to work on the various product model elements assigned to developers
3. [Product delivery] customer needs and product usage scenarios are specified
4. [Product environment] the environment information space and relevant entities are specified
5. [Product requirements] product observable behavior, with quality criteria, is specified
6. [Product delivery] product documentation and training materials for the customer are developed
7. [Product verification] a platform for product testing is developed
8. [Product design] the product's static and physical structures are specified
9. [Product design] the product build platform is developed
10. [Product components] software service and platform components are specified (designed, implemented, and verified)
11. [Product design] the product's dynamic structure is specified
12. [Product components] behavioral components are specified
13. [Product design] a (partial or complete) product version is built
14. [Product verification] the product version is evaluated against scenarios
15. [Product analytics] a completed product version is evaluated against behavioral quality criteria

(any of the above may be iteratively continued until a consistent product model is evaluated as acceptable for customer use)

16. [Product delivery] a verified product is packaged, validated, deployed to the customer, and supported

The objective of such a process is disciplined flexibility that in the end results in a consistent project model. This takes the form of a "concurrent" process such that the order in which activities are performed is not arbitrarily constrained: activities can be performed in a strict "natural" order based on dependencies among them if preferred or in a more relaxed order based on availability of developers with appropriate competence or due to other project circumstances and priorities (e.g., risk mitigation with empirical explorations, analyses of alternatives, addressing customer feedback on prior work). Furthermore, every project model facet has one or more elements that are only weakly dependent on the results of other activities and as such can be independently tasked. There can also be work proceeding concurrently on tasks associated with multiple planned product increments.

The Rationale for a Concurrent Process

Traditional processes typically impose a fixed "natural" ordering on activities (e.g., requirements -> design -> implementation -> test) with the idea that a task cannot be performed until tasks it depends upon have been completed. When a specification is completed prior to its realization being started, the means must exist to provide feedback based on insights or issues discovered in creating the realization, requiring that the specification task be resumed to reestablish consistency between the specification and its realization.

This traditional view originated in imitation of physical assembly line processes but, for software, this view actually works only for simple, well-understood problems having a static solution. This view arose to some degree because it organized a project into "phases" corresponding to a simple ordering of tasks, reducing the complexity of managing a software development project and making both planning and scheduling of work and tracking of progress against a plan simpler. It then compartmentalized information about a product into separate documents associated with each of those phases, typically expressed in the form of a paper or electronic document. Documents become inconsistent when the need arises to change the content of one document but other related documents are not changed accordingly.

A concurrent process allows for more flexible task ordering. Developers may work collaboratively on related tasks to mutually identify and resolve potential conflicts. Similarly, a single developer may be concurrently assigned to simultaneously work on multiple related tasks. Alternatively, a developer working on a task to build a realization can anticipate the guiding specification before its completion and adjust to differences after the specification is complete. Analogously, a specification which logically determines a realization can in fact be influenced by a preceding exploration of alternative realizations based on which of them provides the best resolution of associated tradeoffs.

The motivation for a concurrent product engineering process comes from two perspectives:

- How a sole developer tends to work. A developer working alone to build a product, having no need to coordinate effort with others, is able to perform the equivalent of a concurrent process, continuing or interrupting work associated with any task at any time. Nothing requires that work proceed in a fixed ordering or to completion as understanding of the various elements of an overall solution emerges. To be more precise, tasks are not thought of as being started and stopped but rather are viewed as interleaved and always ongoing with different degrees of attention at any given time. When working on a task that triggers some insight concerning another task, the sole developer can annotate the other task's results with that insight or may even interrupt current work and resume work on the other task. If errors, omissions, or duplications are discovered in preceding results that would affect current work, the developer may correct preceding results before continuing with current work or perform current work based on correct information, only annotating past work to be revised later.
- How conventional development actually works. The traditional image of product development is that work proceeds in a simple prescribed sequence of "phases": requirements defines the problem, design defines the solution, implementation builds the solution, and testing verifies that the solution works. This is the simple image that project management wants to offer as evidence that work is being

> progressively completed. Later phases implicitly include revising results of earlier phases so that the final result will be correct (e.g., performing defect-driven rework of the "completed" implementation as an implicit aspect of the testing phase). A partial "agile" approach has arisen in replacing phases with the rapid iterative repetition of tasks so that all results can be kept mutually consistent as changes are needed. A concurrent process extends this to allow greater flexibility for tasks to be performed in any order as circumstances warrant.

A sequential process can lead to a "stream of consciousness" resolution of alternatives that is captured in in the "wrong" document (e.g., a design is limited by an overly constraining requirements statement based on previous circumstances or arbitrary assumption, or an unstated requirement is decided implicitly by a developer). The issue of making decisions with insufficient information is partially resolved by repeatedly performing the activity sequence. The repeated performance of process activities continues until the targeted product is judged to be sufficiently complete to satisfy customer acceptance criteria without exceeding schedule/funding constraints. Such an ordering is based primarily on information dependencies among activities and assumes that an activity should not be started until there is a completed version of the information its predecessor activities produce.
(fixed ordering leads to information becomes increasingly disorganized; design options become requirements, uncertain requirements become arbitrarily fixed) Each product element is the responsible source for specific information about the product. Different elements may contain related information such that changes in one element require the other to be changed so as to maintain consistency. ... For example, while the requirements model specifies the product's observable behavior, corresponding work on the design and components models may expose omissions or inconsistencies that compel changes to the requirements model. This can have the effect of inverting the conventional notion of dependency between these activities. Based on information dependencies, every activity is both informed by and informing of other activities. When inconsistencies arise between related activities, no assumption can be made as to which activity is correct; simply, appropriate changes must be made that will eliminate

the inconsistencies. A traditional approach would, for example, expect requirements to have been finalized before components can begin to be developed. Often, however, development effort may be reduced if requirements are specified based on an understanding of the capabilities of components built for prior products. Similarly, alternative options for resolving customer needs may be more easily settled if these alternatives can be evaluated in terms of which is more feasible to realize using already existing components. This can lead to a better understanding of how the product will be used or identify issues with the feasibility of realizing the product as specified; these insights can lead in turn to refinements in the requirements, resulting in a product that is a better fit to actual needs. It may be appropriate, for example, to define requirements and develop evaluation criteria concurrently based on independent understandings of customer needs and then use the results of both to inform and align both. Work on related activities can proceed in any order so that all inform all. Precedence is given to consistency of information among activities with the belief that consistent but erroneous information is less detrimental and more easily fixed than the confusion that results from inconsistent information.

A Framework for Developmental Quality

The objective of a development project is to efficiently build an effective product. Productivity, as a measure of the effort a project needs to produce a given product to a specified level of quality, is the degree to which a project achieves this objective. This has two aspects, how much effort is required to produce a minimally acceptable product and how much additional effort is required to achieve a specified higher quality product. By improving productivity, a project can as circumstances warrant better manage the tradeoff between effort and product quality, to either produce a product of acceptable quality with reduced effort or, without increasing the level of effort, produce a higher quality product.

A project can improve its productivity, in the first instance, by defining and improving the process being followed (e.g., by adopting more effective methods or with increased automation of mechanical tasks) and, then, by evaluating their ability to satisfy a program-prescribed set of process quality criteria. The degree to which productivity can

be improved is dependent on developers having a combination of subject matter and technical competence.

As a framework for managing a project's productivity, a notional set of developmental qualities are defined within four overarching categories: feasibility, sustainability, conformity, and verifiability. Depending on the nature of the product being developed and the particular circumstances of the project, the quality factors within each of these categories will vary in their importance, some may not be relevant, and others may need to be added. A project is expected to weigh the importance of every relevant quality factor, the degree to which it is being satisfied, and any options for its improvement, considering tradeoffs among different quality factors and effects on initial and total product lifecycle cost and schedule.

Notional Developmental Qualities 1

Feasibility (or efficacy): the ability to build a product with acceptable behavioral qualities within given time and resource constraints; its possible facets include:

- *Buildability* (or manufacturability) (by balancing among business goals of time to market, cost, and product quality)
- *Deployability* (by being made available for use with acceptable time and effort)
- *Integrability* (by enabling interactions through which both product and external elements are able to operate jointly)
- *Supportability* (by requiring only those resources for operation and support that are consistent with the customer's logistical capabilities)

Sustainability (or modifiability): the ability to change the product to reflect expected changes in needs and operating context; its possible facets include:

- *Maintainability* (by supporting modifications that rectify weaknesses or defects)
- *Configurability* (by supporting modifications that reorganize or reinitialize for changes in behavior)
- *Portability*, (by supporting modifications that permit operating on different computational platforms)
- *Evolvability* (by supporting modifications that address changing needs over its lifecycle while maintaining conceptual integrity)

Notional Developmental Qualities 2

Conformability: the ability to use elements that have been used in building previous products or are anticipated to be usable in building other products; its possible facets include:

- *Irreducibility* (by precluding any behavior that has not been explicitly specified)
- *Modularity* (by consisting of elements that each convey a mutually exclusive and cohesive set of needed capabilities)
- *Adaptability (by* consisting of elements that accommodate being customized or extended in prescribed ways to satisfy anticipated differing (diverse or changing) needs)

Verifiability: the ability to evaluate whether the product satisfies specified criteria; its possible facets include:

- *Readability* (by communicating intended behavior and qualities which appropriately qualified developers can correctly infer from its expressed representation)
- *Observability* (by enabling access to sufficient information (state and rationale) to permit adequate verification and diagnosis of defects)
- *Testability* (by providing means to control, observe, and analyze operation for effects relative to expected behavior and qualities)
- *Certifiability* (by use of practices that are known and can be substantiated as engendering required properties, consistent with legal and regulatory policies and constraints and organizational standards and conventions)