2.0 Basic Software Product Engineering

Software product engineering is the performance of a project to provide a designated simple market with a singular evolving 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 those customers' 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, allowing work on any activity to proceed whenever needed information and resources are available.

A project is defined by a "project model" that specifies both how the project operates and all aspects of the product that results. The project model has three purposes: prescriptive (specifying the operation of the project and the composition of the to-be-built product), predictive (providing a basis for estimating the properties of the project and product), and descriptive (describing the status and rationale for the project and its product as a basis for evaluation and any future development).

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 distinguished elements: a project management model that defines how the project operates and a product model that defines the product that results (Figure 2.0-1). The project model is premised on the competence and conventions imparted to the project by its chartering program and the capabilities needed by its designated customers. 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 project model describes the product from the perspective of the specific developer competence needed for its realization.

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.

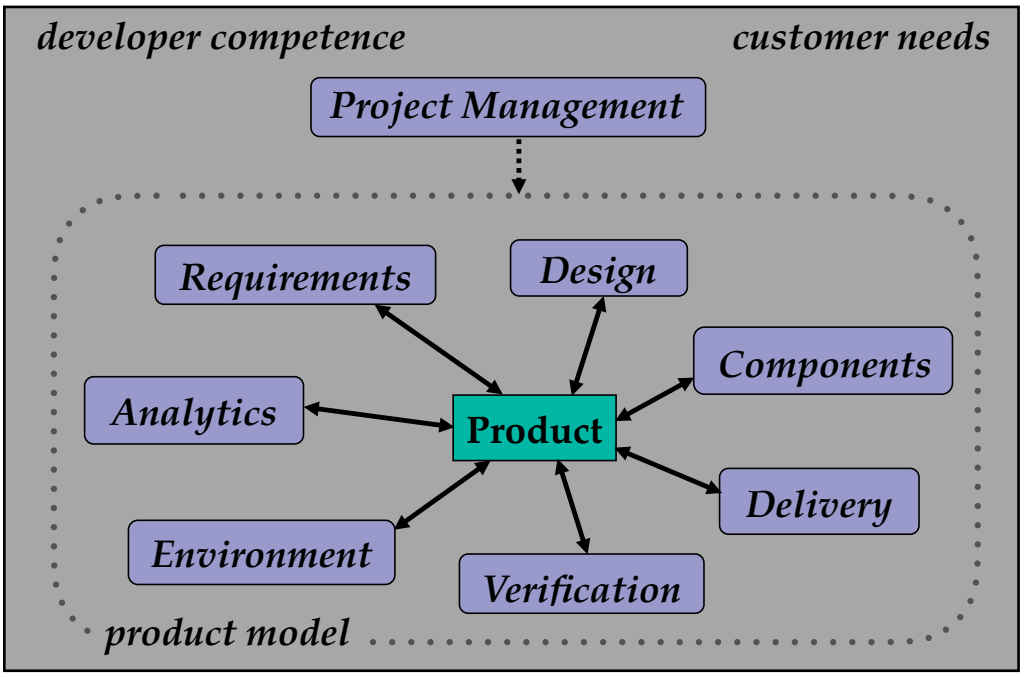

Figure 2.0-1. The Faceted Structure of the Software Project Model

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*** is the aggregate of all materials (documents, specifications, and models) 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.

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 model elements}
{*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/simple market}

- **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 as the subject of software product behavior.
{*objective*: specify the relevant elements and behavior of the operational environment within which the product will operate in support of customer objectives}
{*competence*: how to specify an ecosystem that serves as an information space in which a product can be built to operate}

- **Product Requirements**: A specification of the externally observable behavior that the product is expected to exhibit in operation, consistent with customer-envisioned capabilities and constraints
{*objective*: specify product behavior that provides capabilities supporting customer objectives}
{*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 Design**: A specification of the internal organization of a product and its interactions with its operational environment and its realization in a form that will exhibit requirements-specified behavior
{*objective*: specify the composition and realization of a product that will provide a best approximate satisfaction of behavioral quality criteria}

{*competence*: how to conceive the organization of a product that can be built to provide specified behavior}

- **Product Analytics**: A specification of the quantitative and qualitative analyses used to determine the degree to which a product will satisfy behavioral quality criteria and to understand the implications of problem-solution alternatives
  {*objective*: specify and apply the means to determine the nature and tradeoffs of quality criteria, the effects and implications of these criteria, projected emergent behaviors, and how changes would affect these}
  {*competence*: how to derive estimates of quality factors that a product should exhibit and how to evaluate how alternatives in the product model would affect these factors}

- **Product Components**: A specification of components (fabricated, built, or otherwise obtained) with which a product can be built
  {*objective*: specify the internal design, implementation, and verification of design-specified components}
  {*competence*: how to build or acquire components needed to realize a product, including competence in relevant component subject matter}

- **Product Verification**: A specification of evidence that substantiates the degree to which product model elements for a product are consistent and complete in meeting prescribed product quality criteria, referencing (1) element-associated peer and expert reviews, (2) testing based on scenarios that mimic operational use, and (3) analyses based on application of analytic methods
  {*objective*: specify and apply the means to evaluate the consistency and completeness of the elements of product model elements for a specified product}
  {*competence*: how to realize and apply the means to evaluate a product model and its realization as a product relative to specified behavioral quality criteria}

- **Product Delivery**: A specification of the utility, acceptance, deployment, and support of a verified version of a product for customers
  {*objective*: specify how to frame, prepare, and deploy a product into a customer enterprise in a form that can be employed to the benefit of its operation}

{*competence*: how to work with customers to determine and articulate their operational practices, capability needs, and acceptance criteria, to evaluate, deploy, and support a product for operational use, and to transition customer organizational and individual practices 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 result. 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. 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. (Specific methods are not prescribed for 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, must satisfy.)

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, a simple linear conception of a work flow through those elements is illustrated [Figure 2.0-2]. A realistic work flow is more complexly nuanced with tasks that are more granular, iteratively repeated, and concurrently interwoven. 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

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

Figure 2.0-2. A Simplified Linear Workflow.

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.

The Rationale for a Concurrent Process

Traditional software development processes have typically imposed a fixed "natural" ordering on activities (e.g., requirements -> design -> implementation -> test) with the idea that a task cannot be performed until tasks having content it depends upon have been completed. This view originated in imitation of physical assembly-line processes but, for software, this actually works only for simple, well-understood problems having an easily determined solution. This view simplified the management of software development, enabling an ordering of tasks that was simple to schedule and track progress against a plan. However, in the face of uncertainty and change, this increases the difficulty of maintaining consistency of content when results of later tasks conflict with the results of completed tasks. and tends to disperse related content over the results of multiple tasks, such that complete information is not easily found when needed later.

The objective of an effective process for software product engineering is disciplined flexibility that in the end results in a consistent product model. A "concurrent" process allows for more flexible ordering of tasks in that the order in which activities are addressed 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, identified risks and tradeoffs, or changing customer or project circumstances and priorities.

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 interrupt current work to 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 can offer as evidence that work is being progressively completed. In fact, later phases have always implicitly included revising results of earlier phases so that the final result would be correct (e.g., performing defect-driven rework of the "completed" implementation as an implicit aspect of the testing phase). Incremental and "agile" approaches have arisen to reframe phases for the managed repetition of the conventional linear task ordering to recognize the necessity of revising past results to maintain consistency as work proceeds. A concurrent process extends this to allow greater flexibility for tasks to be performed in any order or combination as circumstances warrant.

Each element of the product model is responsible for defining specific information about the product. Different elements may contain related information; if an element is

changed, related elements may also have to be changed to maintain consistency. For example, changes in the requirements model may require changes in the design or components model. Similarly, work on the design or components models may expose omissions or inconsistencies that compel changes in the requirements model (this can be seen as inverting the conventional notion of dependency between these activities). Based on information dependencies such as these, every activity is both informed by and informing of other activities. Maintaining consistency across all facets of a product model is highly dependent on following an effective version management discipline.

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 inconsistency. Some examples that can reduce development effort are:

- Product requirements and design specified to exploit the capabilities of previously built components;
- Uncertainties about actual customer needs resolved based on the feasibility of realization using components that are being initially built based only on developer projections of what will be needed
- Customer needs and product requirements specified in accordance with preceding empirical product verification results that were based on customer-provided operational scenarios

Work on related activities can proceed in any order so that all are informed by other perspectives. Precedence is given to consistency of information across activities with the belief that inconsistent information is just as detrimental as consistent but erroneous information. There are several ways in which a concurrent process accommodates more flexible task ordering toward this end:

- Every product 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.
- Developers can work collaboratively on related tasks to mutually identify and resolve uncertainties and potential conflicts.

- A single developer may be assigned to combine work on multiple related tasks.
- A developer working on a task to build a realization can anticipate its specification before completion and adjust to differences after the specification is complete.
- An incomplete specification can be influenced by a preceding exploration of potential realizations based on which provides the best resolution of associated tradeoffs.
- At the same time, work can be proceeding on tasks associated with other planned product increments.

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, either to produce a product of acceptable quality with reduced effort or, without increasing the level of effort, to produce a higher quality product.

Any defined process has associated limits on the level of productivity that it supports. Within these limits, productivity depends on developers having a combination of subject matter and technical competence to perform the process as specified. Shortcomings in productivity can be discovered through evaluation of well-defined performance criteria. These shortcomings can be repaired by providing developers with improved guidance and training on good practice. A process that supports higher levels of productivity (e.g., by adopting more effective methods or with increased automation of mechanical tasks) can be defined to provide a more substantial improvement in performance.

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, as detailed below, 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 of 2)

Feasibility (or efficacy): the ability to build a product with acceptable behavioral quality within given time and resource constraints; this can include:

- *Buildability* (or manufacturability) (by balancing 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; this can 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 of 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; this can include:

- *Irreducibility* (by excluding any element content that provides redundant or unneeded capability)
- *Modularity* (by including elements that each convey a mutually exclusive and cohesive set of needed capabilities)
- *Adaptability* (by including 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; this can 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)