2.7 Product Components

The product components model specifies the realizations of the components described in the software architecture static view of the product design model. These components, with which the product is built, must be realized in conformance with all elements of the product design model. (A realization of a software component is referred to as a "module" whereas a realization of a hardware component is referred to as a "device".) A component is represented by three elements: a specification (of its interface and internal design), an implementation (software code that realizes the specification), and a verification (substantiation that the implementation conforms to the specification). A component may be realized by multiple modules that may represent differences in the component specification.

Each module implementation can created as a crafted artifact or acquired from a separate, project-designated repository of previously built components. In either case, the implementation must conform to the module specification and satisfy the module verification. For each module, the product design (in the static view dependency relation) defines which other components it depends on for its implementation; a module to be used must be composed with those other modules that it requires (a prebuilt module may be self-contained, in which case it may include functionality that follows different conventions than that provided by similar crafted modules).

Dependencies

collaborative: do components get what they need from other components so they can satisfy the product requirements and design

Module Specification

The module specification element defines the responsibilities of a software component, consistent with product requirements and in conformance with the component's definition as prescribed in the product design. Component responsibilities take the form either of functionality that the component enacts to meet requirements or services it provides for use in the implementation of other components.

<----------

differing approaches based on component type:

(hdw hiding: encapsulate w sw + device + device emulation for any device with which the sw interacts)

(beh hiding: beh according to Prod Rqmts)

(sw services: according to needs of other components)

---------->

A module specification has two parts, a module interface specification and a module internal design specification. A module interface specification describes the facilities needed for other components to interact with the component, either controls for managing the component's functionality or services that other components can request. A module internal design specification describes how the module is implemented and associated rationale.

Module Interface Specification

(how to use, facilities provided (name, purpose, parameters, exceptions), quality factors, ?)

Module Internal Design Specification

(assumptions, trace to rqmts/design, analysis of alternatives considered, potential changes, extensions/unimplemented (or excluded) features, instrumentation features)

Module Implementation

The module implementation element specifies the computational realization of a module. As a general rule, the implementation of a module is expressed in a human-understandable "source" form that is either mechanically translated by means of a well-defined transformation into a computer-understandable "object" form or processed at runtime by a mechanism that interprets its meaning in terms of equivalent hardware operations. In either case, a module is typically only usable through combination with other modules upon whose services it depends.

A module implementation in either form may also be obtained from a repository of separately created modules. Such implementations can be used as-is, customized using an associated mechanism, or uniquely tailored. In all cases, the implementation must either conform to or provoke changes in the module specification.

In building a new product, developers reasonably tend to consider starting with a copy of all or parts of a more-or-less similar, previously-built product. Such opportunistic reuse of past results can save effort if developers can be sure that assumptions and tradeoffs upon which the development of those reused parts was based are appropriate for the new product. In too many cases, trying to reuse parts built for a different purpose will be found to be usable for a new product only with significant effort to discover and revise aspects that differ. Practical, cost-effective reuse depends on creating assets with a clear definition of the conditions under which proper reuse can occur: what differing aspects can be reasonably changed and how, and what aspects would be difficult to change. Consequently, software reuse as traditionally practiced is not well suited to building multiple products for a market in which customers have some degree of diversity in their needs. Whether a product built for one customer is a suitable basis for building a similar product for another customer is not easily determined after the prior product has been built if appropriate consideration was not given to how differences in needs between customers are to be addressed.

A component implementation consists of both source material and associated annotations that (1) explain all primary elements of the constructed source and (2) provide supplemental insights that may inform future modifications.

(coding form conventions, instrumentation for runtime observability/introspection-explanation {may be defined by optionally used service components})

(creating component mockup for other hdw or sw component testing)

(instrumentation for verif/val or runtime data collection/analyses)

Module Verification

The module verification element specifies the means used to establish that a module as designed and implemented conforms to relevant aspects of product requirements and product design specifications, including completeness, and consistency with module interface specifications of supporting components. Means used to verify a module include peer reviews, analytic evaluations, and testing.

Peer Reviews

(describe how to orchestrate these) (developmental qualities re sw architecture, related components, testing; behavioral qualities)

Static Analyses

(qualitative & quantitative measures) (automated eg lint, compilation, product subset builds, ?)

Product Subset Builds

(build of product subset sufficient to perform testing of this module; instrumented for observability) {using Product Delivery mech to build according to Product Design Physical model}

Testing and Dynamic Analyses

(dynamic analytics, test case scenarios/data, automated regression testing, replication with adaptable tests, ?) specify and apply analytics and verification practices to module

emulate platform & ecosystem (using openv) & hdw, run tests based on developer scenarios & eval results