## 2.7 Product Components

The product components model specifies the components, as defined in the product design model, that constitute the content with which a product is built. A component has three elements: a specification (of its interface and internal design), a realization (as a set of discriminated implementations), and a verification (substantiation that the implementations conform to the specification).

(A “module” is a particular software implementation of a component. A module may encapsulate a related set of physical or emulated entities.)

A component may have alternate realizations as modules corresponding to differences in specific component responsibilities and how these are actualized (for example, a module responsible for interactions with an operational entity versus another that emulates interactions with such an entity, or modules that differ in providing faster versus more accurate computation).

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 component specification and satisfy the module verification. For each component, 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).

### Component Specification

The component 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.

***Interface Specification***

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

***Internal Design Specification***

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

## Component Implementation

The component implementation element specifies the computational realization of one or a set of modules that each satisfy the component (interface and internal design) specification.

As a general rule, the implementation of a constructed component instance is expressed in a human-understandable "source" form that is either "compiled" (mechanically translated by means of a well-defined transformation into a computer-understandable "object" form) or "interpreted" (processed at runtime by a mechanism that interprets its meaning in terms of equivalent hardware operations). In either case, a module requires access to the services of any other modules on which it depends.

A component 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 prompt changes in the component specification.

(The ability to reuse components, particular those having alternative implementations of component responsibilities, are discussed further in section 4.1.) 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.

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)

## Component Verification

The component verification element specifies the means used to establish the degree to which the component specification and its implementation(s) conform in realizing the component's responsibilities in the product design, including consistency with the interface specification of supporting components. Means used to verify a module include peer and expert reviews, analytic evaluations, and testing.

### *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, ?)

### *Dependencies*

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

### *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} {build sw wrapper as root component to wrap component for testing}

### *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