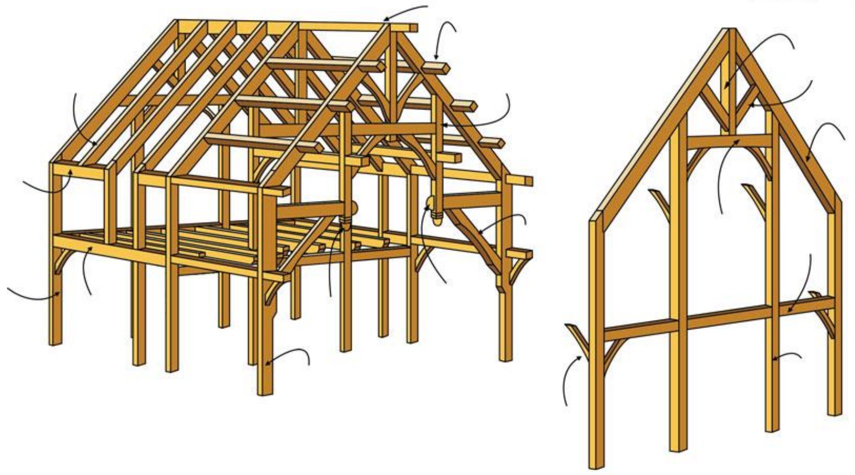
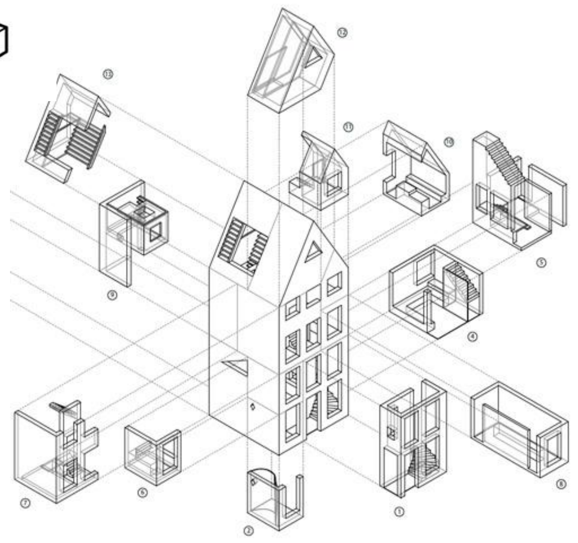
Software Product Line Engineering

Lab Class 7 / Assignment 4+5

Task 1a)

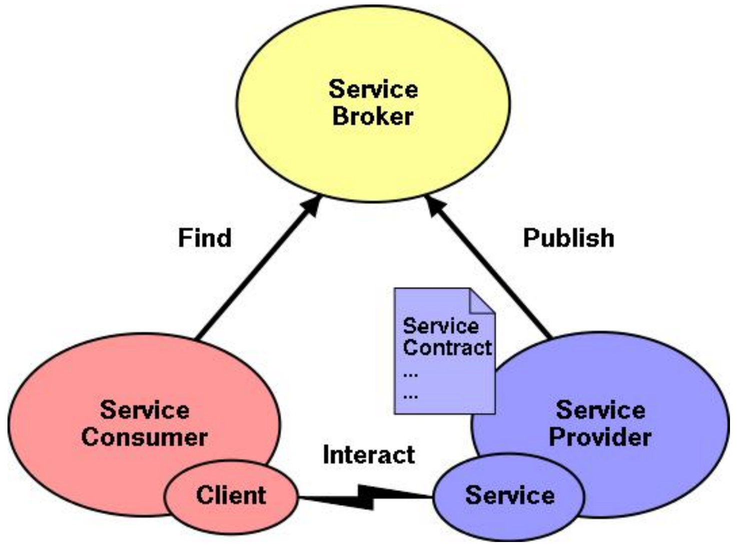




Software Component
- Cohesive, modular implementation unit with an interface
- Encapsulate functionality
- Composed with other components

Software Framework
- Reusable solution for a family of problems in a domain
- Abstract structure that can be tailored for specific use case
- Functionality extensible at hot spots

SOA
- Encapsulate functionality (service)
- Distributed scenarios (e.g., web)
- Composition using orchestration



Task 1b)

For **white-box frameworks**, we require *knowledge of internals* to extend it via overriding and addition of methods.

Use cases: good code knowledge, reuse not top priority

For **black-box framework**, behavior is added using components via *provided interfaces*.

Use cases: focus on reusability, component usage

Task 1c)

For **white-box frameworks**, we use structural design patterns based on **inheritance**:

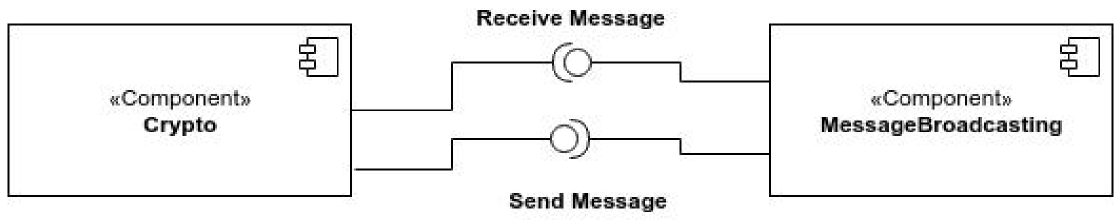
Template Method Pattern (overriding method in subclass)

For **black-box framework**, we use structural design patterns based on **interfaces**:

Strategy Pattern (alternative strategy provided by subclass)
Observer Pattern (cf. plugin loading)

Task 1d) Components and Services

- Components/Services ~ features
- Components are composed and connected manually by glue code.



Task 1e) Components and Services

+	-
• Features are clearly *modularized*.	• *Runtime overhead* introduced by glue code.
• Independently developed components ease *collaboration*.	• No *automated product derivation*.
• Easy to incorporate *third-party components*.	• Optimal component size hard to estimate (*preplanning problem*)
• Dependencies are expressed via glue code.	• *Retrospective modifications* of the framework and component model are problematic.

Task 1f) Components and Services

	Proactive SPL development	Reactive SPL development	Extractive SPL development
Components /SOA	• Features are modularized, but glue code necessary for derivation.	• *Retrospective introductions or modifications* of the framework and component model are problematic	
White-Box Frameworks	• Automated product derivation. • Features are modularized.		
Black-Box Frameworks			

Task 1f) Components and Services

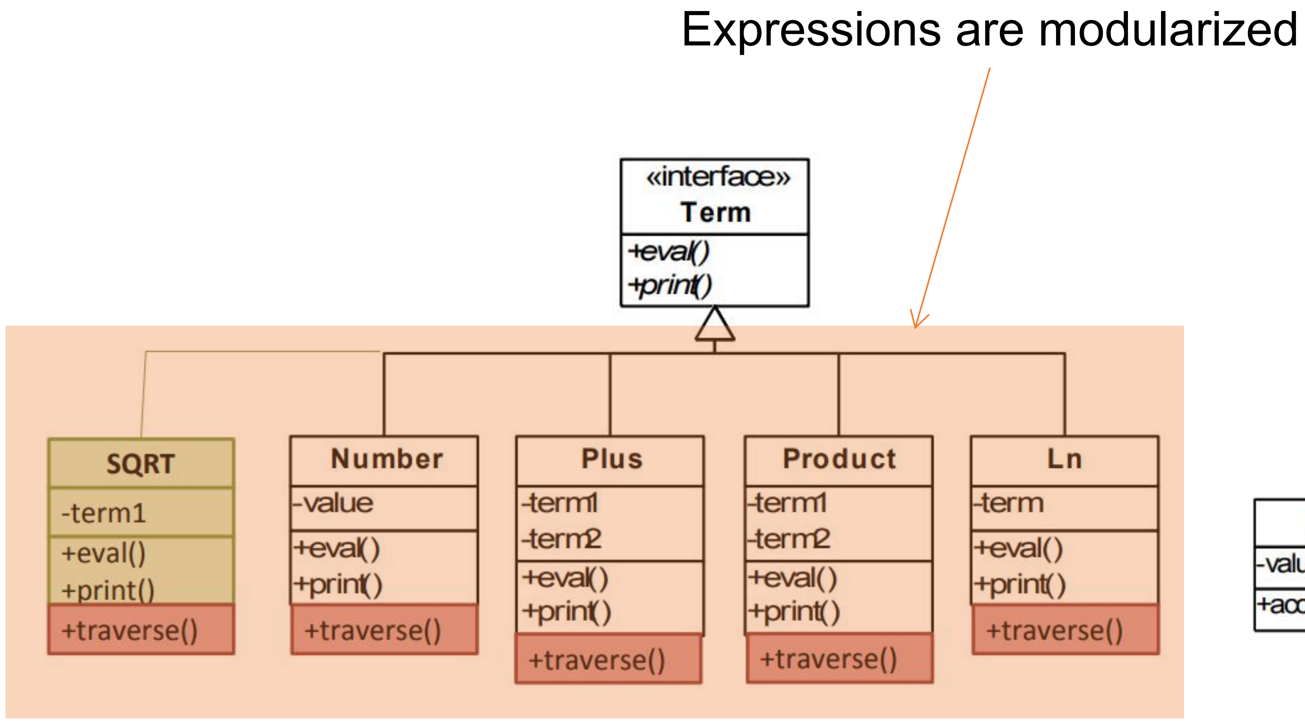
	Embedded Systems	Required Skills	Scalability
Components /SOA	• Runtime overhead by glue code (for components) and unnecessary code shipped with each variant	• Only required to understand the component model.	• Features are modularized. • Multiple features can be combined.
White-Box Frameworks		• Deep understanding of the framework implementation is required.	
Black-Box Frameworks		• Only required to understand a number of interfaces.	

Task 2a) Cross-Cutting Concerns

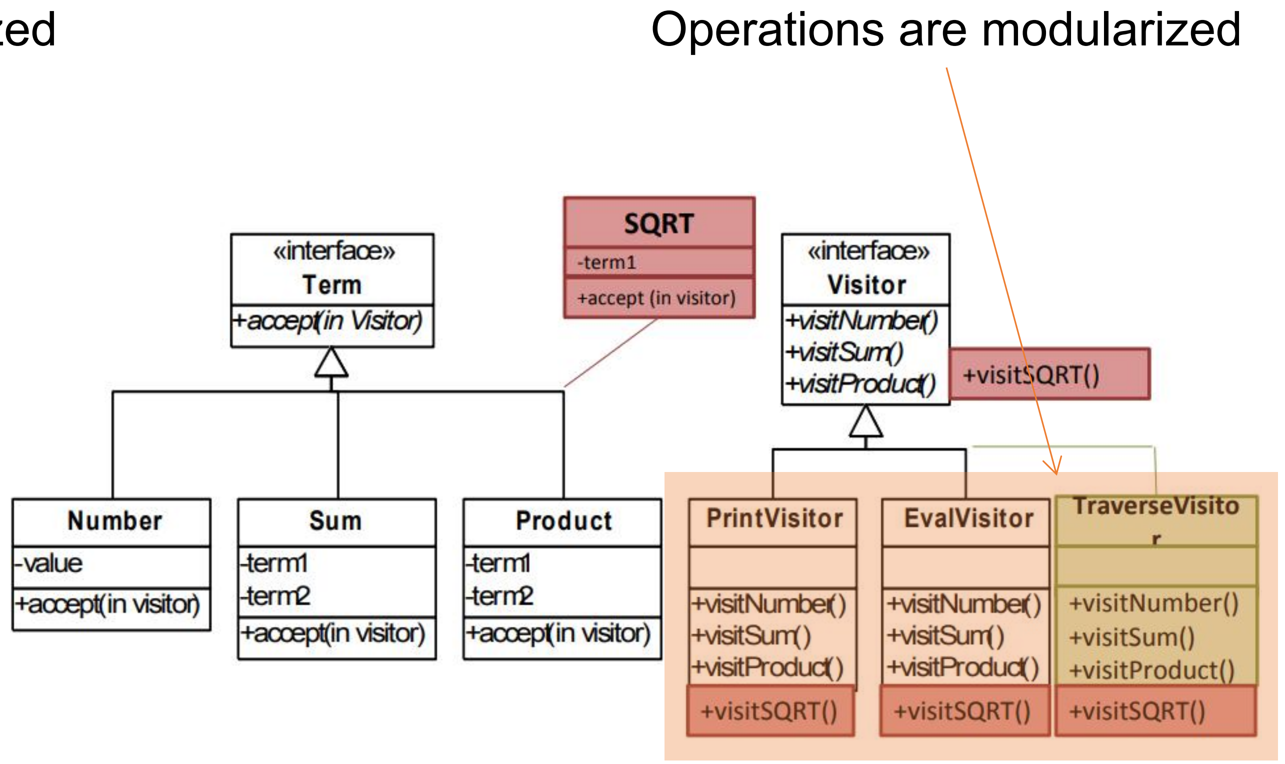
- Concerns ~ features
- Cross-cutting concerns are concerns that cannot be or are not modularized, such as Logging.

Task 2a) Tyranny of the Dominant Decomposition

"Many concerns can be modularized, but not at the same time. Simultaneous modularization along different dimensions not possible"



Expressions modularized (data-centric)



Operations modularized (method-centric)

Task 2b+c) Preplanning Problem

b) To keep an SPL extensible, extensibility must be pre-planned! Developers must anticipate and procactively provide extension points to avoid changing base later.

c) Complex class-subclass structures are sensitive to changes ot the code base. In frameworks, it is best to not instantiate directly, but modularize instantiation in a separate place (factory).