Systems Engineering

Lecture 12

Reuse in Systems Engineering

Dr. Joanna Bryson
Dr. Leon Watts

University of Bath
Department of Computer Science

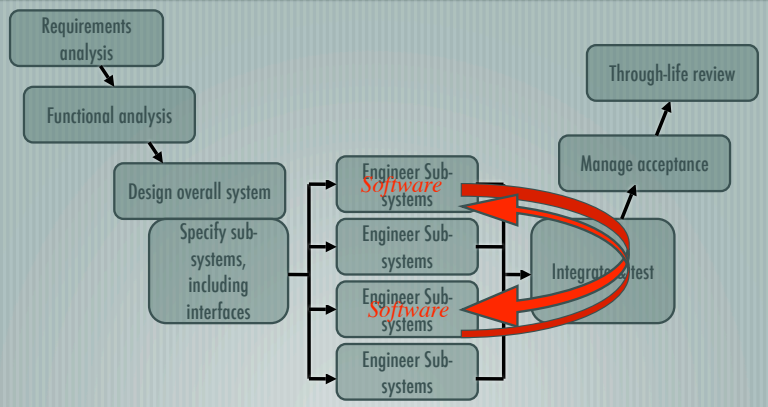
Learning outcomes

- After attending this lecture and doing the reading, you should be able to:
  - Explain why reuse is important in systems engineering.
  - Define the term "Reuse Landscape" with examples.
  - Discuss the benefits and limitations of reuse in a given project.
  - Give examples of four different types of reuse in SE.
  - Describe component based reuse and COTS software development, and discuss their relative advantages and limitations.

Reusing knowledge in systems engineering

- Systems engineering projects generally involve engineers from different disciplines.
  - Different disciplines work with a different knowledge. base
  - Want to re-use knowledge and practices.
  - Software re-design typically compensates for misunderstandings earlier in process.

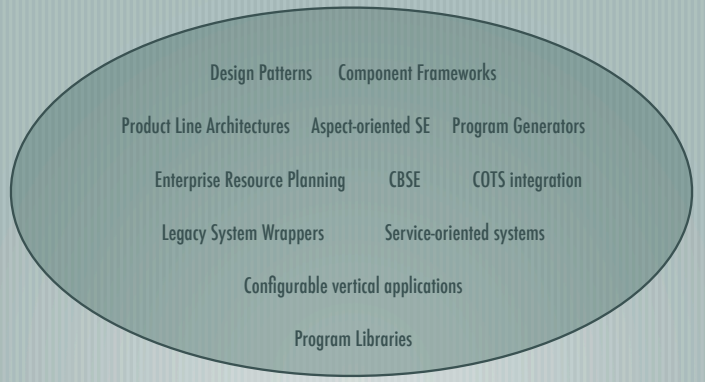
Systems Engineering V



System Engineering 'V' Process

- Sub-systems are generally developed in parallel.
  - Defining sub-system interfaces is a critical activity for parallel sub-system development; changes are costly.
- Limited scope for iteration between phases.
  - Software is often used to compensate for problems with hardware design.
- Identifying sub-systems makes use of knowledge of existing components.
  - Need to match and assign requirements to sub-systems.

The Reuse Landscape



Reuse Benefits

- SE companies report benefits from in-house reuse:
  - Fujitsu : 20% to 70% more projects completed on-time
  - NEC : 7-fold productivity increase
  - HP : 40% reduction in time to market and approx. 35% fewer faults
- Achieved by years of investment in reuse
  - Designing software to be reusable (more expensive).
  - Investment in specialised reuse repositories for component retrieval and demand assessment.

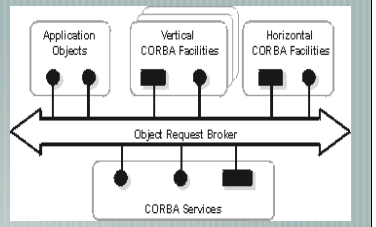
Software Reuse

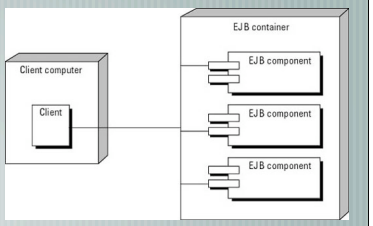
- SE companies are increasing their software reuse
  - Component dependability and process risk
  - Encapsulation of specialist knowledge
  - Compliance with standards (e.g. UI)
  - Accelerated development schedule
  - Shift to software-as-a-service (SOA)
- Most common types of software reuse:
  - Component Reuse (CBSE)
  - Application Reuse (COTS Integration)
  - Objects / Functions (OO Reuse)
  - Concept reuse (e.g. Design Patterns)

} The "Reuse landscape"

Example Component Models

- A component model is a definition of standards for component implementation, documentation and deployment.
- Common Object Request Broker Architecture (CORBA) component model
  - 'Middleware' allowing integration of many different machines and technologies.
- Enterprise Java Beans (EJB) component architectural model
- COM+ model (.NET model)





Component Based Reuse

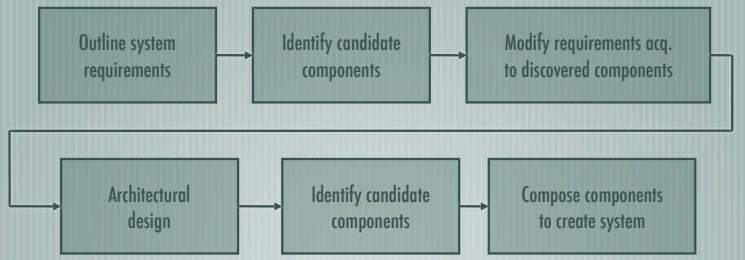
- Object and function reuse market is disappointing
  - Too fine grain
  - Component based reuse focus on medium-grain software units
- What is a component?
  - "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties" [Szyperski '02]
- Component characteristics
  - Standardisation (component model)
  - Composable
  - Independent (provides / requires interface)
  - Documented (API)
  - Deployable

Developing Components for Reuse

- Component reusability
  - Should reflect stable domain abstractions;
  - Should hide state representation;
  - Should be as independent as possible;
  - Should publish exceptions through the component interface.
- Developing components for reuse can be expensive
  - Components must be specially developed for reuse.
  - The more general the interface, the greater the reusability but more complexity = less reusability.
  - Legacy systems are often "wrapped" for reuse to save on the cost of rewriting them.

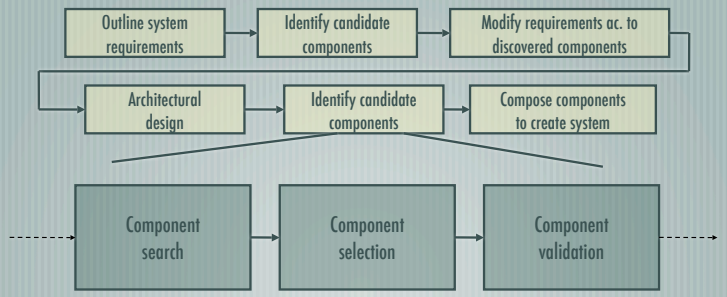
Component based reuse (CBSE)

- CBSE Process model:
  - for predominantly reuse oriented development.



Component based reuse (CBSE)

- CBSE Process model:
  - for predominantly reuse oriented development.

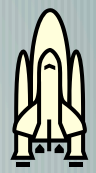


Issues with CBSE

- Need to be able to trust the component supplier.
  - At best, an untrusted component may not operate as advertised; at worst, it can breach your security.
- Requirements
  - Different groups of components will satisfy different requirements (conflicts)
- Validation & Verification
  - The component specification may not be detailed enough to allow comprehensive tests to be developed.
  - Components may have unwanted functionality. How can you test this will not interfere with your application?

Component Reuse Failure Case

- Ariane 5 launcher
  - Reuse of inertial reference software from Ariane 4
    - Black box reuse
    - Field-proven code
  - Rocket de-stabilised and was forced to self-destruct only 37 seconds into launch.
  - Software had shut down at runtime.
    - Overflow exception in numeric conversion routine.
    - Rocket engines too powerful.
  - Routine was not used in Ariane 5 mission.
    - No requirement so no test.



Application (COTS) reuse

- Coarse grain reuse.
- Greatly accelerated development times.
  - Potentially at the cost of maintainability.
- Software product lines (in house).
  - Risks variations on your product due to subsequent market changes.
- Third party vendors.
  - E.g. databases linked to report generators.

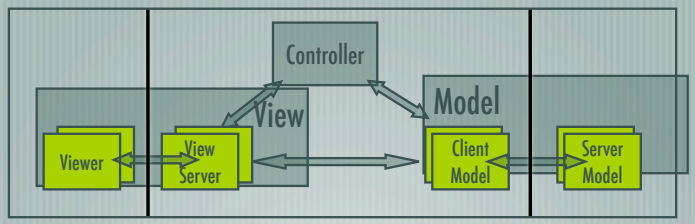
Potential Pitfalls of COTS Development

- Lack of control over functionality and performance.
  - COTS systems may be less effective than they appear.
- Problems with COTS system inter-operability.
  - Different COTS systems may make different assumptions that means integration is difficult.
- No control over system evolution.
  - COTS vendors not system users control evolution.
- Support from COTS vendors.
  - COTS vendors may not offer support over the lifetime of the product.

White Box Reuse

- Source code availability (and documentation) greatly improves maintainability and aids verification.
  - Open source development
  - Licensing
- Generator-based reuse (a form of concept reuse)
  - Application generators for business data processing.
  - Parser and lexical analyser generators for language processing.
  - Code generators in CASE tools.
    - Consider the cost of understanding / maintaining the code.
  - Very cost-effective but only possible in a relatively small number of application domains.

Combining Component Standards



Disadvantages of Reuse

- Verification implications for your system.
  - "Black box" re-use: Certification and liability of 3rd party code.
  - Emergent properties.
- Restrictions on requirements and evolution.
- Reuse repository.
  - Populating with components can be expensive.
  - CASE support may be poor.
  - Search (component discovery).
- Distrust within the development team.
  - NEC: "In most firms, each department prefers to use only its own piece of code to execute a function, and does not trust code developed elsewhere."

Summary

- Reuse is important in systems engineering because concurrent development and integration of sub-systems is essential.
- The “Reuse Landscape” covers a wide range of elements and methods for their combination.
- Reuse can speed up development but externally sourced may compromise requirements.
- Now know examples of four types of reuse common in software engineering and their relative advantages and limitations.