Chapter 20 – Systems of Systems

Topics covered

- ✧ System complexity
- ✧ System of systems classification
- ✧ Reductionism and complex systems
- ✧ Systems of systems engineering
- ✧ Systems of systems architecture

Systems of systems

- More and more systems are being constructed by integrated existing, independent systems
- *A system of systems is a system that contains two or more independently managed elements.*
- There is no single manager for all of the parts of the system of systems and that different parts of a system are subject to different management and control policies and rules.

Examples of systems of systems

- A cloud management system that handles local private cloud management and management of servers on public clouds such as Amazon and Microsoft.
- An online banking system that handles loan requests and which connects to a credit reference system provided by credit reference agency to check the credit of applicants.
- An emergency information system that integrates information from police, ambulance, fire and coastguard services about the assets available to deal with civil emergencies such as flooding and large-scale accidents.

Essential characteristics of SoS

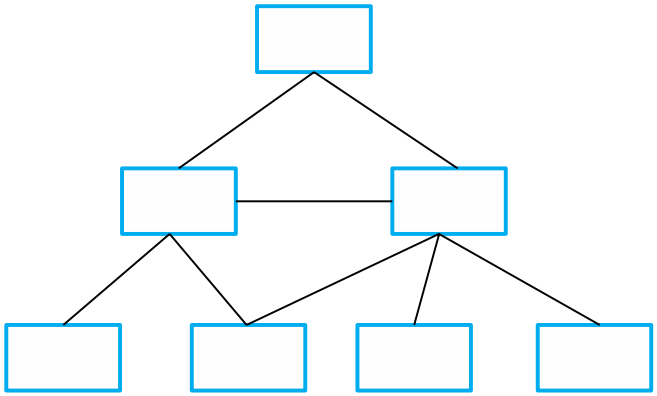
- ✧ Operational independence of system elements
- ✧ Managerial independence of system elements
- ✧ Evolutionary development
- ✧ Emergence of system characteristics
- ✧ Geographic distribution of system elements
- ✧ Data intensive (data >> code)
- ✧ Heterogeneity

System complexity

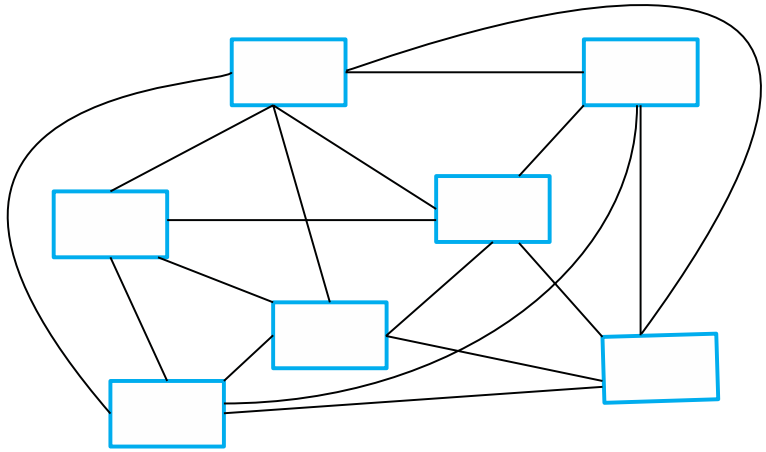
Complexity

- All systems are composed of parts (elements) with relationships between these elements of the system.
  - For example, the parts of a program may be objects and the parts of each object may be constants, variables and methods.
  - Examples of relationships include ‘calls’ (method A calls method B), ‘inherits-from’ (object X inherits the methods and attributes of object Y) and ‘part of’ (method A is part of object X).
- The complexity of any system depends on the number and the types of relationships between system elements.
- The type of relationship (static or dynamic) also influences the overall complexity of a system.

Simple and complex systems



System (a)

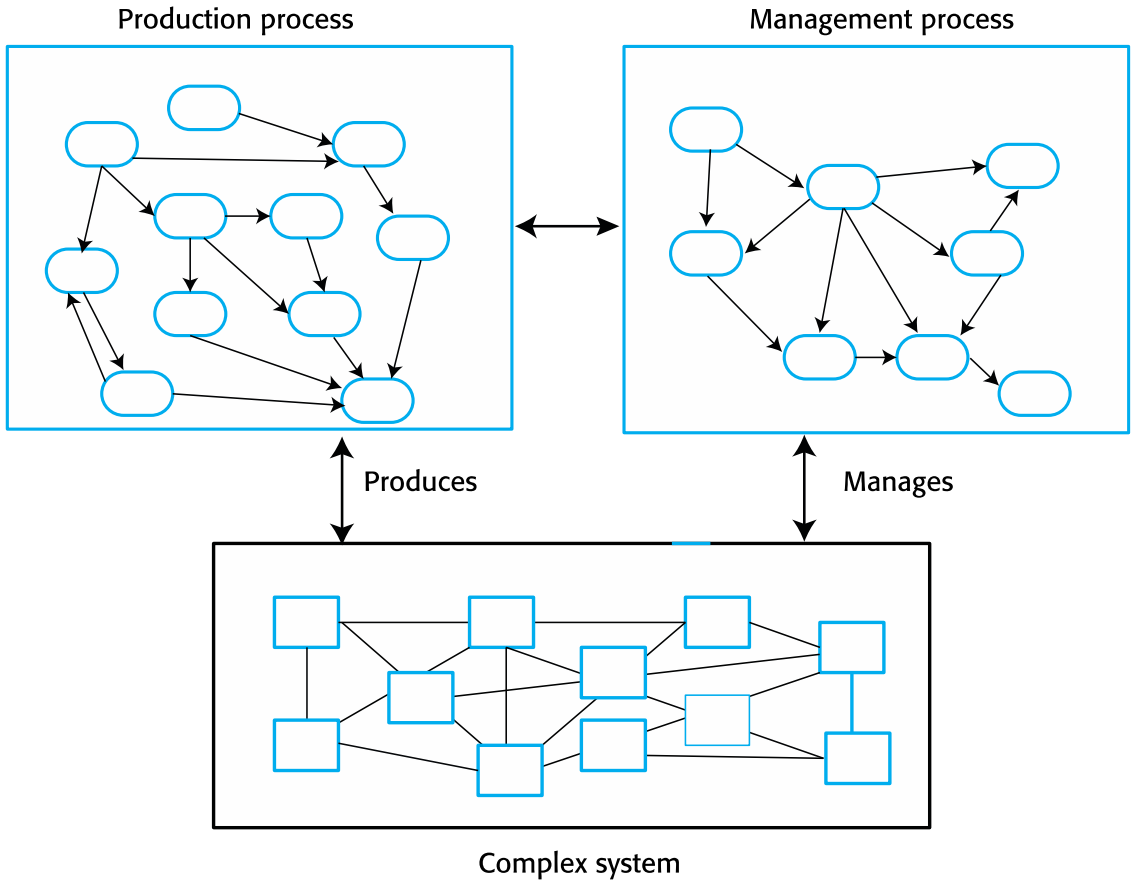


System (b)

Process complexity

- As systems grow in size, they need more complex production and management processes.
- Complex processes are themselves complex systems.
  - They are difficult to understand and may have undesirable emergent properties. They are more time consuming than simpler processes and they require more documentation and coordination between the people and the organizations involved in the system development.
- The complexity of the production process is one of the main reasons why projects go wrong, with software delivered late and over-budget.

System production and management processes



Complexity and software engineering

- Complexity is important for software engineering because it is the main influence on the understandability and the changeability of a system.
- The more complex a system, the more difficult it is to understand and analyze.
- As complexity increases, there are more and more relationships between elements of the system and an increased likelihood that changing one part of a system will have undesirable effects elsewhere.

Types of complexity

- *Technical complexity* is derived from the relationships between the different components of the system itself.
- M*anagerial complexity* is derived from the complexity of the relationships between the system and its managers and the relationships between the managers of different parts of the system.
- *Governance complexity* of a system depends on the relationships between the laws, regulations and policies that affect the system and the relationships between the decision-making processes in the organizations responsible for the system.

System characteristics and complexity

SoS characteristic	Technical complexity	Managerial complexity	Governance complexity
Operational independence		X	X
Managerial independence	X	X	
Evolutionary development	X		
Emergence	X		
Geographical distribution	X	X	X
Data-intensive	X		X
Heterogeneity	X		

Complexity and project failure

- Large-scale systems of systems are now unimaginably complex entities that cannot be understood or analyzed as a whole.
- The large number of interactions between the parts and the dynamic nature of these interactions means that conventional engineering approaches do not work well for complex systems.
- It is complexity that is the root cause of problems in projects to develop large software-intensive systems, not poor management or technical failings.

Systems of systems classification

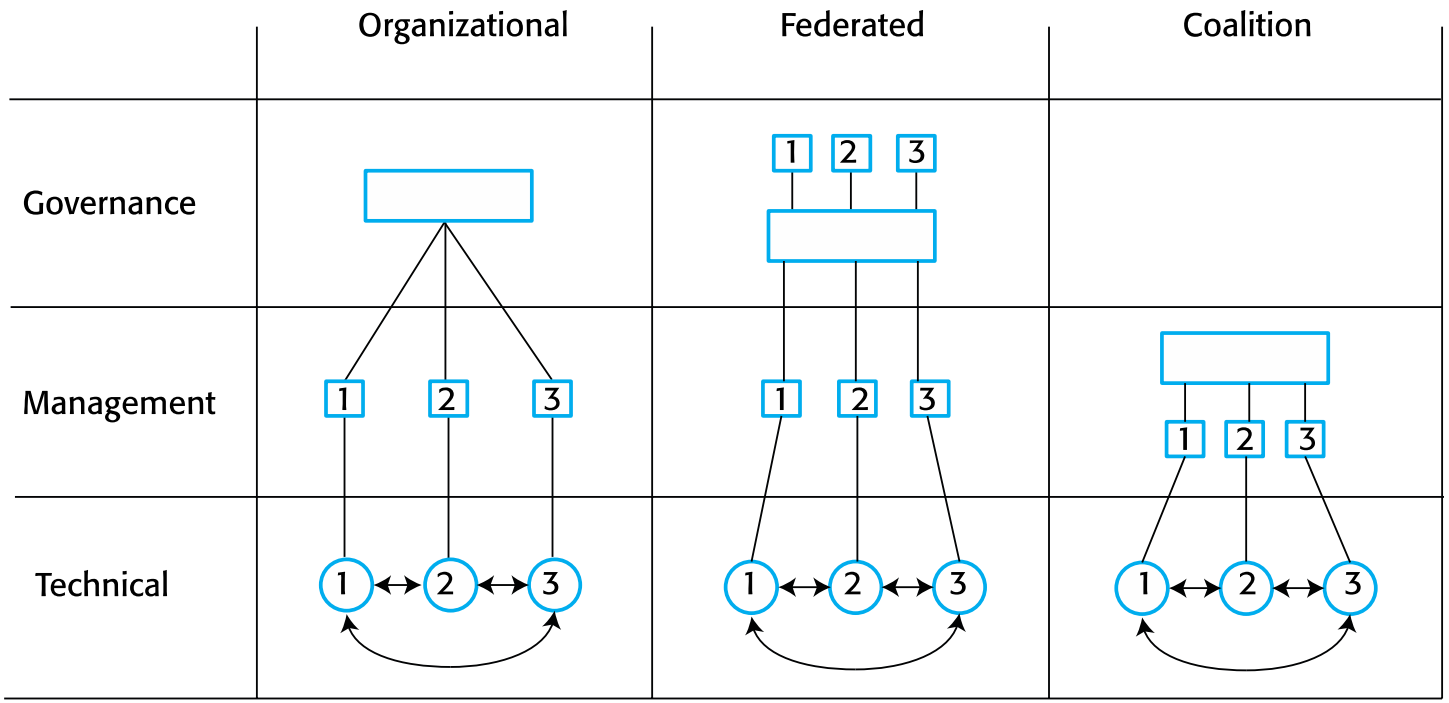
Maier's classification of systems of systems

- **Directed SoS** are owned by a single organization and are developed by integrating systems that are also owned by that organization. The system elements may be independently managed by parts of the organization.
- **Collaborative SoS** are systems where there is no central authority to set management priorities and resolve disputes. Typically, elements of the system are owned and governed by different organizations.
- **Virtual systems** have no central governance and the participants may not agree on the overall purpose of the system. Participant systems may enter or leave the SoS.

More intuitive classification terms

- *Organizational systems of systems* are SoS where the governance and management of the system lies within the same organization or company.
- *Federated systems* are SoS where the governance of the SoS depends on a voluntary participative body in which all of the system owners are represented.
- *System of system coalitions* are SoS where there are no formal governance mechanisms but where the organizations involved informally collaborate and manage their own systems to maintain the system as a whole.

System of systems classification



iLearn as a SoS

- iLearn is a relatively simple technical system but it has a high level of governance complexity.
- The development of a digital learning system is a national initiative but to create a digital learning environment, it has to be integrated with network management and school administration systems.
- There is no common governance process across authorities so, according to the classification scheme, this is a coalition of systems.

Reductionism and complex systems

Complexity management in engineering

- The approach that has been the basis of complexity management in software engineering is called *reductionism*.
- Reductionism is based on the assumption that any system is made up of parts or subsystems.
  - It assumes that the behaviour and properties of the system as a whole can be understood and predicted by understanding the individual parts and the relationships between these parts.
- To design a system, the parts making up that system are identified, constructed separately and then assembled into the complete system.

Software engineering methods

- A reductionist approach has been the basis of software engineering for almost 50 years.
- Top-down design, where you start with a very high-level model of a system and break this down to its components is a reductionist approach.
  - This is the basis of all software design methods, such as object-oriented design. Programming languages include abstractions, such as procedures and objects that directly reflect reductionist system decomposition.
  - Agile methods are also reductionist. The difference between agile methods and top-down design is that system decomposition is incremental when an agile approach is used.

Reductionist methods

- Reductionist methods are successful when there are relatively few relationships between the parts of a system and it is possible to model these relationships.
- Software engineering methods attempt to limit complexity by controlling the relationships between parts of the system.
- Reductionism does not work well when there are many relationships in a system and when these relationships are difficult to understand and analyze.
  - The fundamental assumptions that are inherent to reductionism are inapplicable for large and complex systems

Reductionist assumptions

- *System ownership and control*
  - Reductionism assumes that there is a controlling authority for a system that can resolve disputes and make high-level technical decisions that will apply across the system.
- *Rational decision making*
  - Reductionism assumes that interactions between components can be objectively assessed by, for example, mathematical modelling.
- *Defined system boundaries*
  - Reductionism assumes that the boundaries of a system can be agreed and defined.

System of systems reality

Control

Owners of a system control its development

Decision making

Decisions are made rationally, driven by technical criteria

Problem definition

There is a definable problem and clear system boundaries

Reductionist assumptions

There is no single system owner or controller

Decision-making driven by political motives

Wicked problem with constantly renegotiated system boundaries

Systems of systems reality

Reductionism and software SoS

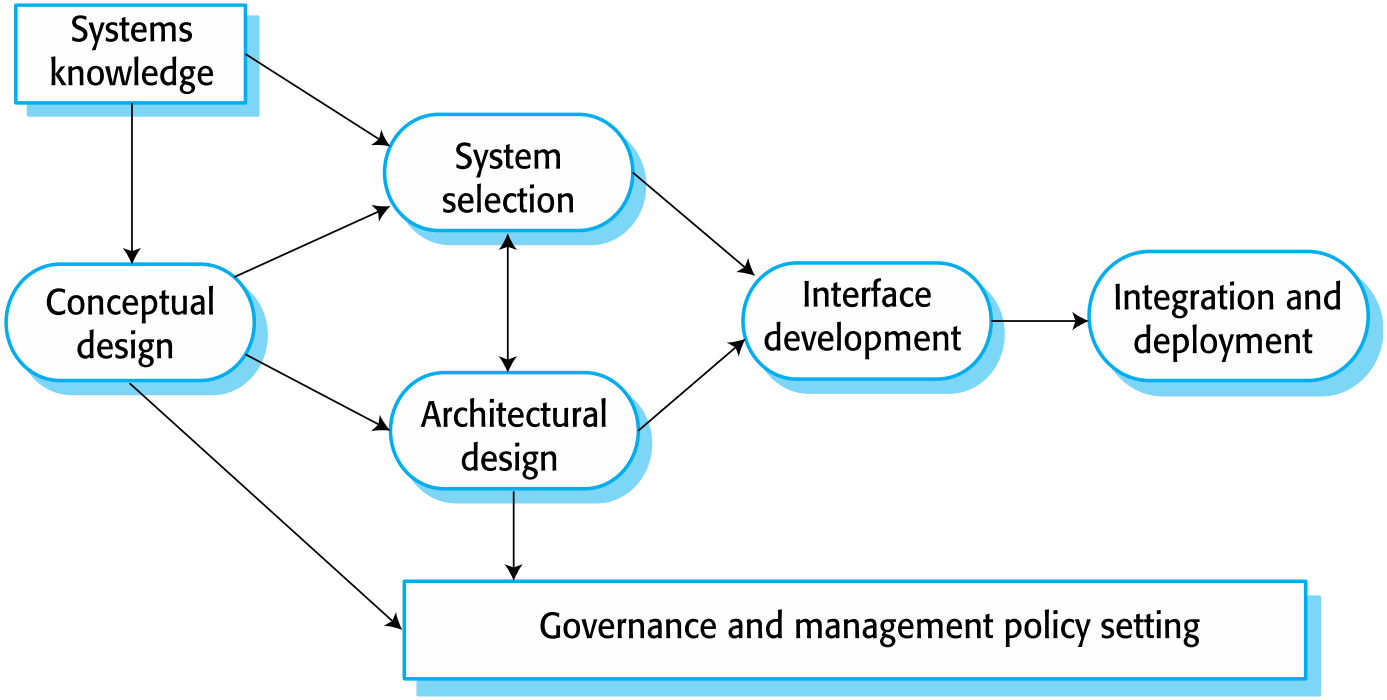
- Relationships in software systems are not governed by physical laws.
  - Political factors are usually the driver of decision making for large and complex software systems.
- Software has no physical limitations hence there are no limits on where the boundaries of a system are drawn.
  - The boundaries and the scope of a system are likely to change during its development.
- Linking software systems from different owners is relatively easy hence we are more likely to try and create a SoS where there is no single governing body.

Systems of systems engineering

SoS engineering problems

- Lack of control over system functionality and performance.
- Differing and incompatible assumptions made by the developers of the different systems.
- Different evolution strategies and timetables for the different systems.
- Lack of support from system owners when problems arise.

Systems of systems engineering



SoS development processes

- *Conceptual design* is the activity of creating a high-level vision for a system, defining essential requirements and identifying constraints on the overall system.
- *System selection,* where a set of systems for inclusion in the SoS is chosen.
  - Political imperatives and issues of system governance and management are often the key factors that influence what systems are included in a SoS.
- *Architectural design* where an overall architecture for the SoS is developed.

SoS development processes

- *Interface development* – the development of system interfaces so that the constituent systems can interoperate.
- *Integration and deployment* – making the different systems involved in the SoS work together and interoperate through the developed interfaces.
- System deployment means putting the system into place in the organizations concerned and making it operational.

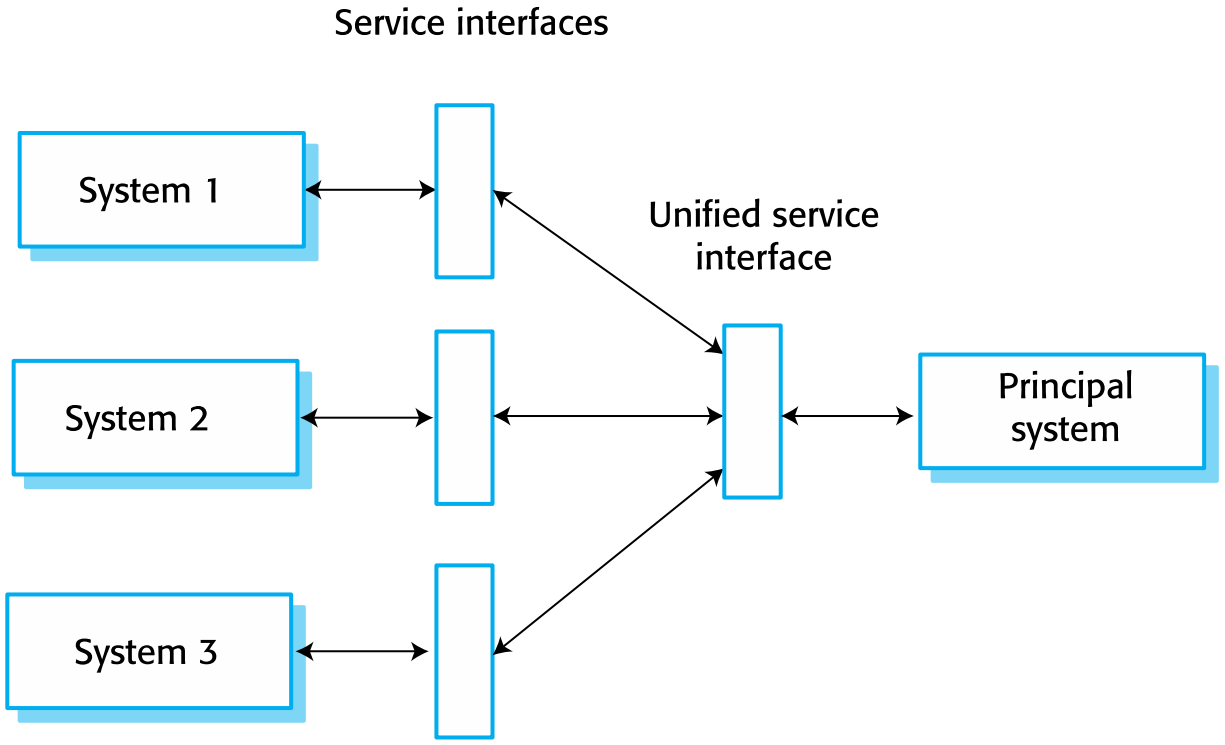
Interface development

- In general, the aim in SoS development is for systems to be able to communicate directly with each other without user intervention.
- Service interfaces
  - If systems in a SoS have service interfaces, they can communicate directly via these interfaces
- The constituent systems in a SoS often have their own specialized API or only allow their functionality to be accessed through their user interfaces.
  - You therefore have to develop software that reconciles the differences between these interfaces.

Service interface development

- To develop service-based interfaces, you have to examine the functionality of existing systems and define a set of services to reflect that functionality.
- The services are implemented either by calls to the underlying system API or by mimicking user interaction with the system.
- A principal system acts as a service broker, directing service calls between the different systems in the SoS.
- Each system therefore does not need to know which other system is providing a called service.

Service interfaces



Unified user interfaces

- User interfaces for each system in a SoS are likely to be different.
- A principal system must have some overall user interfaces that handles user authentication and provides access to the features of the underlying system.
- It is usually expensive and time-consuming to implement a unified user interface to replace the individual interfaces of the underlying systems.

Cost-effectiveness of UI development

- The interaction assumptions of the systems in the SoS
  - If systems have different interaction models, unifying these in a single UI is very difficult
- The mode of use of the SoS
  - A unified UI slows down interaction if most of the interaction is with a principal system in the SoS
- The 'openness' of the SoS
  - If the SoS is open, so that new systems may be added to it when it is in use, then unified UI development is impractical.

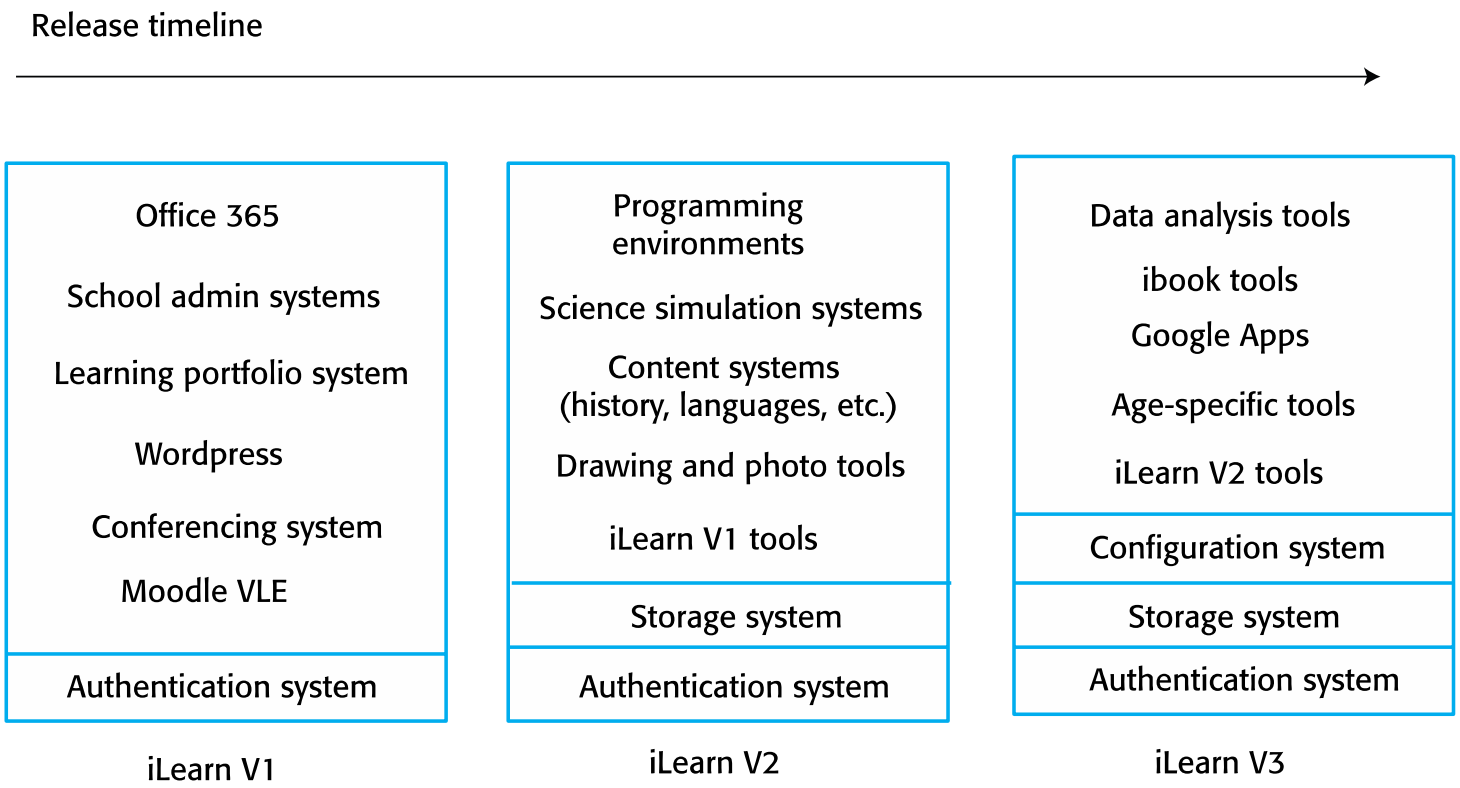
Integration and deployment

- For SoS, it makes sense to consider integration and deployment to be part of the same process.
- Separate integration may be difficult as some of the systems in the SoS may already be in use
- The integration process should begin with systems that are already deployed, with new systems added to the SoS to provide coherent additions to the functionality of the overall system.

Staged deployment of the iLearn system

- The initial deployment provides authentication, basic learning functionality and integration with school administration systems.
- Stage 2 adds an integrated storage system and a set of more specialized tools to support subject-specific learning.
- Stage 3 adds features for user configuration and the ability for users to add new systems to the iLearn environment.

iLearn releases



SoS testing

- There are three reasons why testing systems of systems is difficult and expensive:
  - There may not be a detailed requirements specification that can be used as a basis for system testing. It may not be cost effective to develop a SoS requirements document – the details of the system functionality are defined by the systems included.
  - The constituent systems may change in the course of the testing process so tests may not be repeatable.
  - If problems are discovered, it may not be possible to fix the problems by requiring one of more of the constituent systems to be changed. Intermediate software may have to be introduced to solve the problem.

SoS testing and agile testing

- Agile methods do not rely on having a complete system specification for system acceptance testing.
- Stakeholders are engaged with the testing process and to decide when the overall system is acceptable.
- For SoS, a range of stakeholders should be involved in the testing process if possible and they can comment on whether or not the system is ready for deployment.
- Agile methods make extensive use of automated testing. This makes it much easier to rerun tests to discover if unexpected system changes have caused problems for the SoS as a whole.

Systems of systems architecture

General principles for architecting SoS

- ✧ Design systems so that they can deliver value if they are incomplete.
- ✧ Be realistic about what can be controlled.
- ✧ Focus on the system interfaces.
- ✧ Provide collaboration incentives.
- ✧ Design a SoS as node and web architecture.
- ✧ Specify behaviour as services exchanged between nodes.
- ✧ Understand and manage system vulnerabilities.

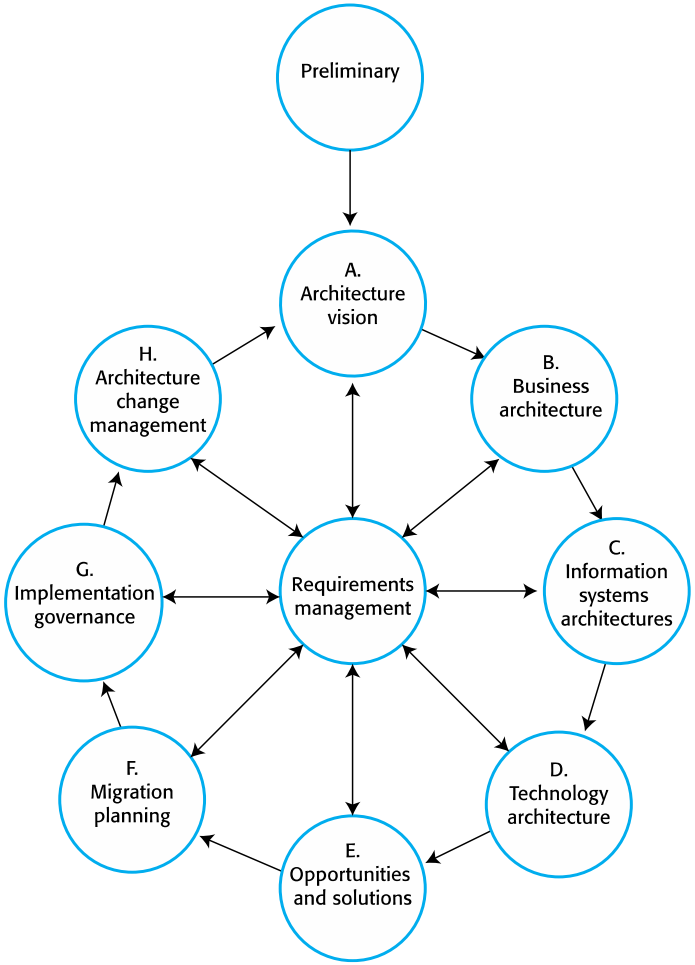
Architectural frameworks

- Architectural frameworks such as MODAFand TOGAF have been suggested as a means to support the architectural design of systems of systems.
- An architecture framework recognises that a single model of an architecture does not present all of the information needed for architectural and business analysis.
- Frameworks propose a number of architectural views that should be created and maintained to describe and document enterprise systems.

TOGAF

- The TOGAF framework has been developed by the Open Group as an open standard and is intended to support the design of a business architecture, a data architecture, an application architecture and a technology architecture for an enterprise.
- At its heart is the Architecture Development Method (ADM), which consists of a number of discrete phases.

TOGAF – Architecture Development Method



Architectural model management

- Initial model development takes a long time and involves extensive negotiations between system stakeholders. This slows the development of the overall system.
- It is time-consuming and expensive to maintain model consistency as changes are made to the organization and the constituent systems in a SoS.

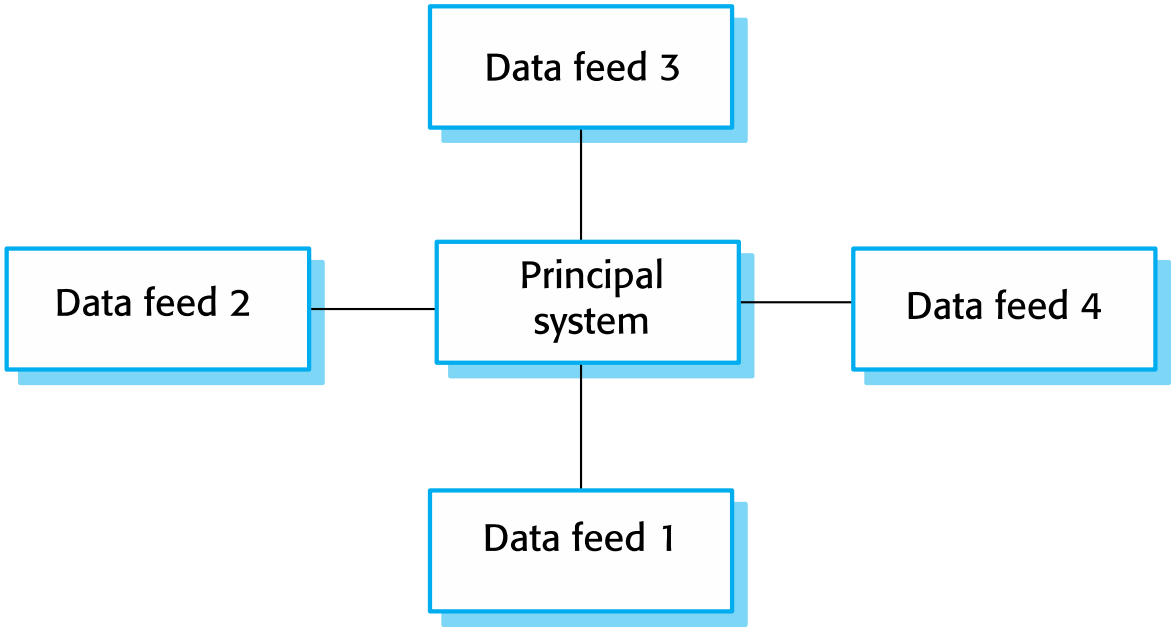
Architectural patterns for SoS

- An architectural pattern is a stylized architecture that can be recognized across a range of different systems.
- Architectural patterns are a useful way of stimulating discussions about the most appropriate architecture for a system and for documenting and explaining the architectures used.

Systems as data feeds

- There is a principal system that requires data of different types.
- This data is available from other systems and the principal system queries these systems to get the data required.
- Generally, the systems that provide data do not interact with each other.
- This pattern is often observed in organizational or federated systems where some governance mechanisms are in place.

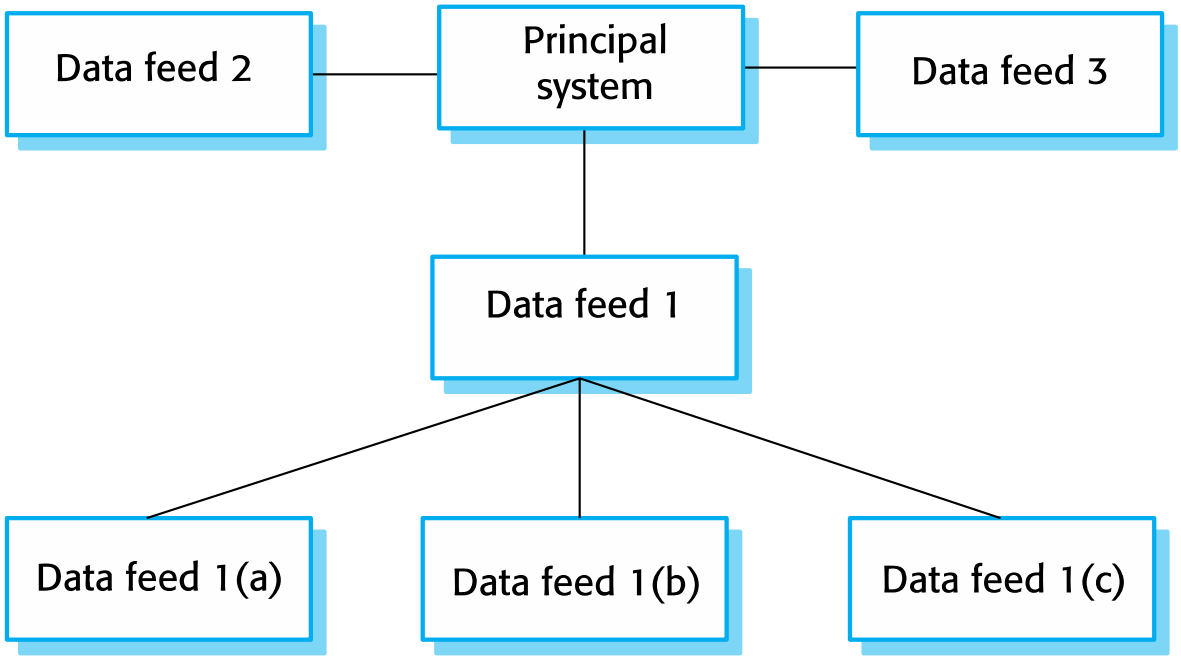
Systems as data feeds



Systems as data feeds

- The ‘systems as data feeds’ architecture is an appropriate architecture to use when it is possible to identify entities in a unique way and create relatively simple queries about these entities.
- A variant of the ‘systems as data feeds’ architecture arises when there are a number of systems involved which provide similar data but which are not identical.
- The architecture has to include an intermediate layer to translate the general query from the principal system into the specific query required by the individual information system.

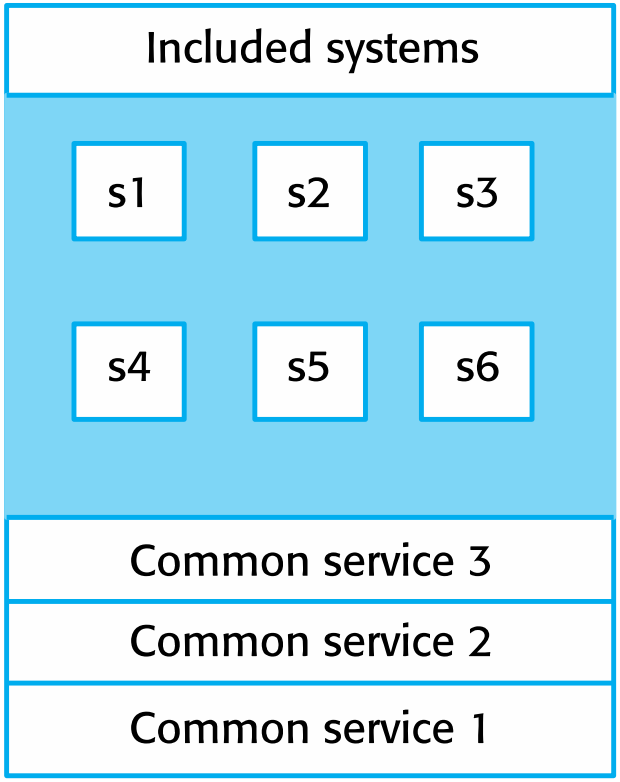
Systems as data feeds with unifying interface



Systems in a container

- Systems in a container are systems of systems where one of the systems acts as a virtual container and provides a set of common services such as an authentication and a storage service.
- Conceptually, other systems are then placed into this container to make their functionality accessible to system users.
- You don’t place systems into a real container to implement these systems of systems. Rather, for each approved system, there is a separate interface that allows it to be integrated with the common services.

Container systems

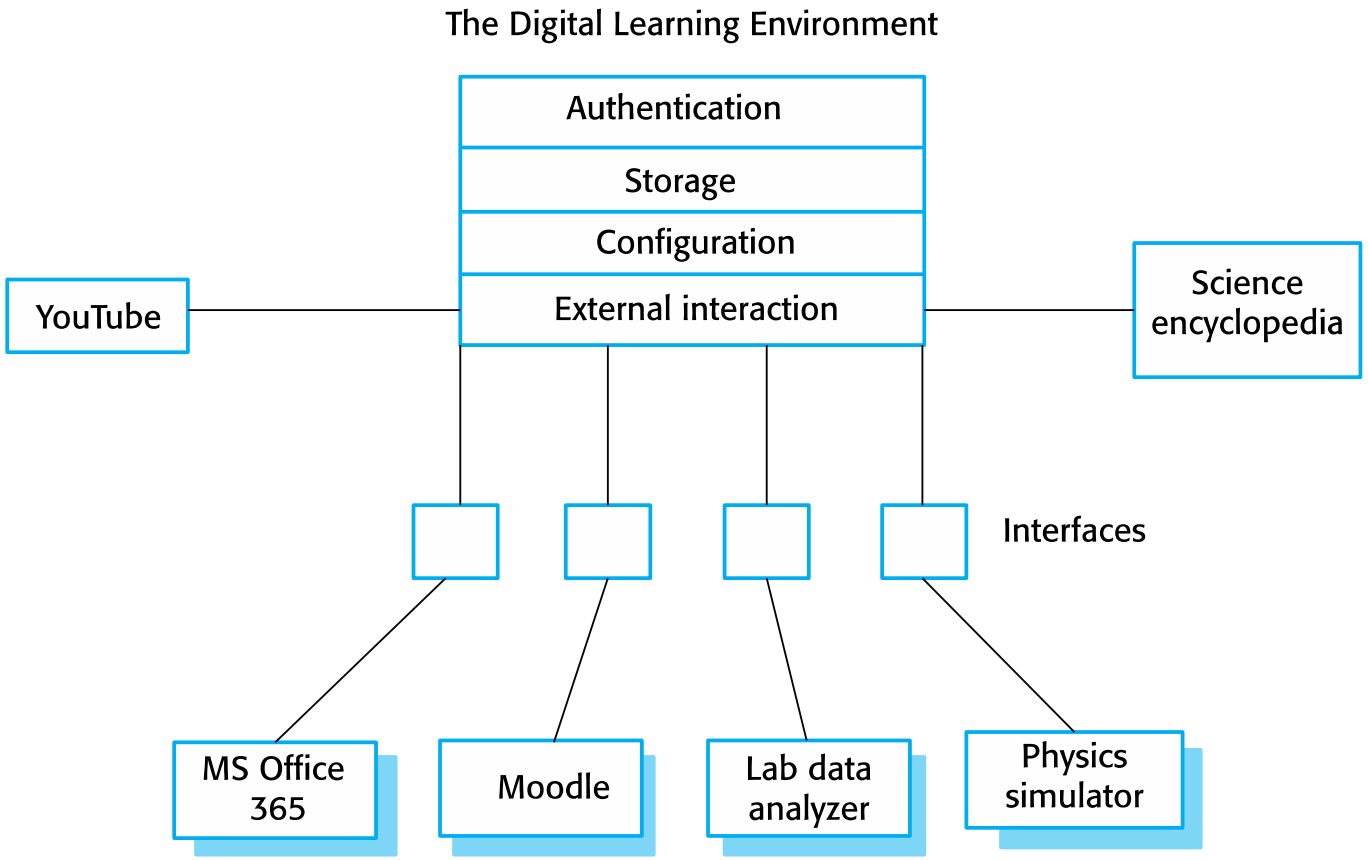


Container system

ILearn container: common services

- An authentication service that provides a single sign-in to all approved systems. Users do not have to maintain separate credentials for these.
- A storage service for user data. This can be seamlessly transferred to and from approved systems.
- A configuration service that is used to include or remove systems from the container.

iLearn as a container



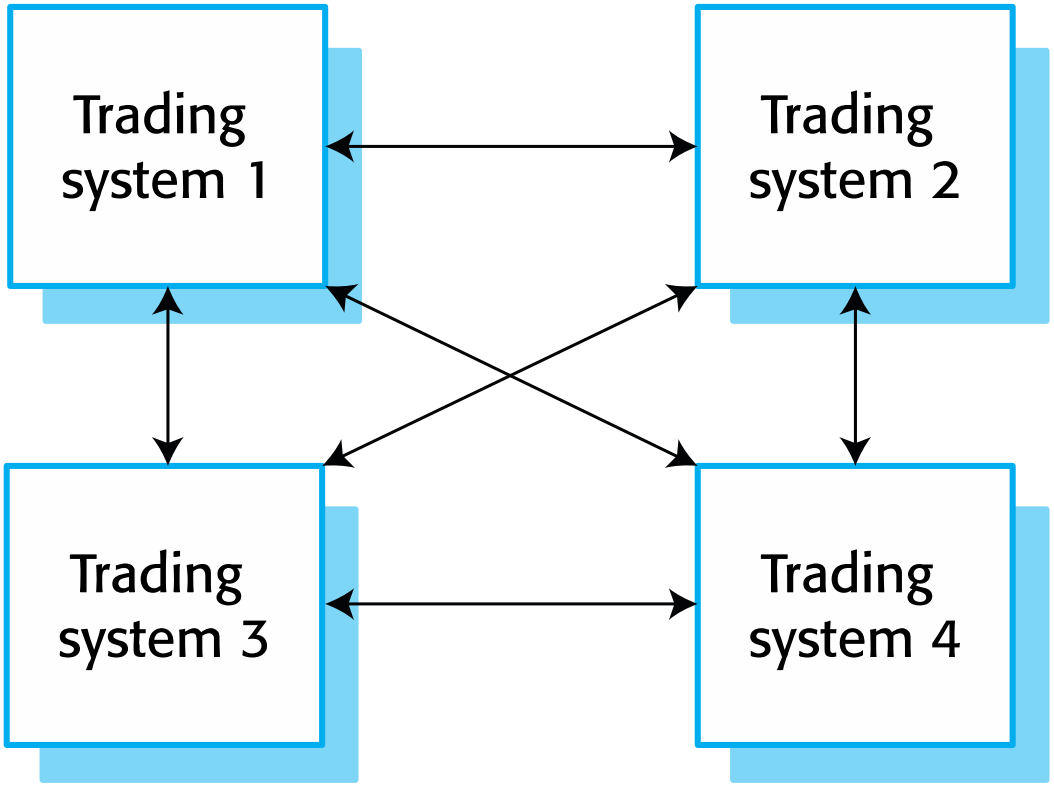
Container architecture problems

- A separate interface must be developed for each approved system so that common services can be used with these systems.
- This means that only a relatively small number of approved systems can be supported.
- The owners of the container system have no influence on the functionality and behaviour of the included systems. Systems may stop working or may be withdrawn at any time.

Trading systems

- Trading systems are systems of systems where there is no single principal system but processing may take place in any of the constituent systems.
- The systems involved trade information amongst themselves. There may be one-to-one or one-to-many interactions between these systems.
- Each system publishes its own interface but there may not be any interface standards that are followed by all systems.

Trading systems



Trading SoS

- Trading systems may be developed for any type of marketplace with the information exchanged being information about the goods being traded and their prices.
- While trading systems are systems in their own right and could conceivably be used for individual trading, they are most useful in an automated trading context where the systems negotiate directly with each other.
- The major problem with this type of system is that there is no governance mechanism so any of the systems involved may change at any time.

Key points

- Systems of systems are systems where two or more of the constituent systems are independently managed and governed.
- There are three types of complexity that are important for systems of systems – technical complexity, managerial complexity and governance complexity.
- System governance can be used as the basis for a classification scheme for SoS. This leads to three classes of SoS namely organizational systems, federated systems and system coalitions.

Key points

- Reductionism as an engineering method breaks down because of the inherent complexity of systems of systems.
- Reductionism assumes clear system boundaries, rational decision making and well-defined problems. None of these are true for systems of systems.
- The key stages of the SoS development process are conceptual design, system selection, architectural design, interface development and integration and deployment. Governance and management policies must be designed in parallel with these activities.

Key points

- Architectural patterns for systems of systems are a means of describing and discussing typical architectures for SoS.
- Important patterns are systems as data feeds, systems in a container and trading systems.