Chapter 20 – Systems of Systems

1

Topics covered

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

2

Systems of systems

- The increase in size of software systems is remarkable – today's large systems may be a hundred or thousand times larger than the "large" systems of the 1960s.
- Very large-scale systems now and in the future will be built by integrating existing systems from different providers to create systems of systems (SoS).
- *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 different parts of a system are subject to different management and control policies and rules.

3

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 agencies 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.

4

Essential characteristics of SoS

- *Operational independence of elements* Parts of the system are not simply components but can operate as useful systems in their own right.
- *Managerial independence of elements* Parts of the system are "owned" and managed by different organizations or by different parts of a larger organization. This is the key factor that distinguishes a system of systems from a system.
- *Evolutionary development* SoS are not developed in a single project but evolve over time from their constituent systems.

5

Essential characteristics of SoS

- *Emergence* SoS have emergent characteristics (e.g. security, reliability, volume, etc.) that only become apparent after the SoS has been created.
- *Geographical distribution of elements* The elements of an SoS are often geographically distributed across different organizations.
- *Data intensive* A software SoS typically relies on and manages a very large volume of data.
- *Heterogeneity* The different systems in a software SoS are unlikely to have been developed using the same programming languages and design methods.

6

System complexity

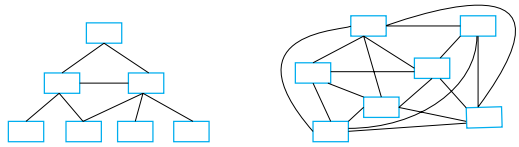
7

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 such as a "uses" one or dynamic such as a "calls" one involved in, say, if-statements) also influences the overall complexity of a system.

8

Simple and complex systems



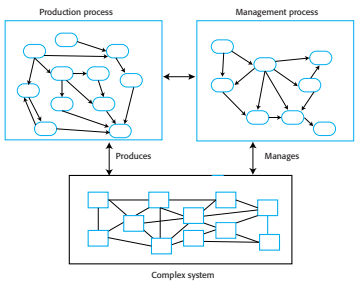
9

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.

10

System production and management processes



11

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.

12

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.

13

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		

14

Examples of SoS characteristics vs. type of complexity

- *Operational independence* Constituent systems are subject to different rules/policies (governance complexity) and ways of managing the system (managerial complexity).
- *Managerial independence* Management changes should be consistent (managerial complexity) and special support software for these changes may be needed (technical complexity).
- *Evolutionary development* Different parts of the SoS may be built using different technologies (technical complexity).

15

Examples of SoS characteristics vs. type of complexity

- *Emergence* The more complex a system, the more likely it is that it will have undesirable emergent properties (technical complexity).
- *Geographical distribution* Software is required to coordinate/synchronize remote systems (technical complexity), managers based in different countries have difficulty coordinating their actions (managerial complexity), and different parts of the SoS may be located in different jurisdictions (governance complexity).
- *Data intensive systems* Need to cope with data errors and incompleteness (technical complexity) and the existence of different laws (governance complexity).
- *Heterogeneity* Difficulties in ensuring compatibility between different technologies used in different parts of the SoS (technical complexity).

16

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.

17

Systems of systems classification

18

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.

19

More intuitive classification terms (governance-based classification)

- *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.

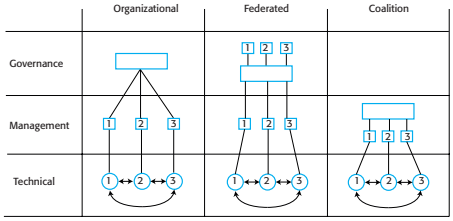
20

Examples of SoS, based on the different classification schemes

- A military command-and-control system that integrates information from airborne and ground-based systems is an example of a directed or organizational SoS.
- An integrated public transport information system is an example of a collaborative or federated SoS. Bus, rail, and air transport providers agree to link their systems to provide passengers with up-to-date information.
- An example of a virtual SoS (or one based on coalitions) is an automated high-speed algorithmic trading system. These systems from different companies automatically buy and sell stock from each other, with trades taking place in fractions of a second.

21

System of systems classification



22

iLearn as an SoS

- iLearn, a case study in the book, is a system that provides a range of learning support by integrating separate software systems such as Microsoft Office 365, virtual learning environments such as Moodle, simulation modeling tools, and content such as newspaper archives.
- 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.

23

Reductionism and complex systems

24

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.

25

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.

26

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.

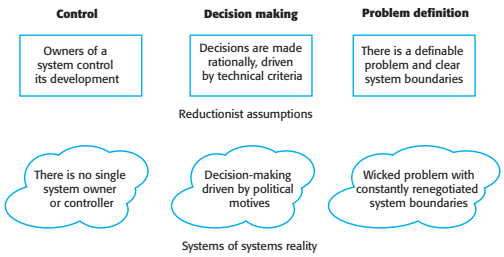
27

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.

28

System of systems reality



29

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 an SoS where there is no single governing body.

30

Systems of systems engineering

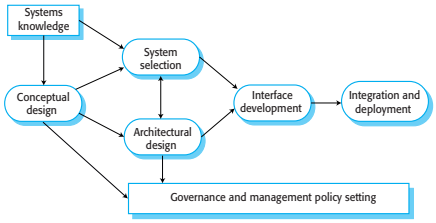
31

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.

32

A systems of systems engineering process



33

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. An important input here is knowledge of the existing systems that may participate in the SoS.
- *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 an SoS.
- *Architectural design*, where an overall architecture for the SoS is developed.

34

SoS development processes

- *Interface development* The development of system interfaces so that the constituent systems can interoperate. This may also involve the development of a unified UI so that SoS operators do not have to deal with multiple user interfaces as they use the different systems in the SoS.
- *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.

35

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 an SoS have service interfaces, they can communicate directly via these interfaces.
- The constituent systems in an 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.

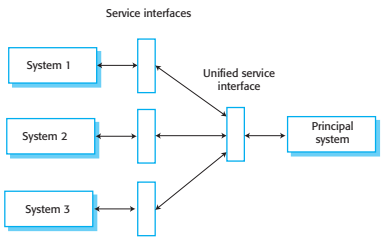
36

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.

37

Service interfaces



38

Unified user (service) interfaces

- User interfaces for each system in an 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.

39

Cost-effectiveness of UI development

- The interaction assumptions of the systems in the SoS
  - Some systems may have a process-driven model of interaction where the system controls the interface and prompts the user for inputs. Others may give control to the user, so that the user chooses the sequence of interactions with the system.
  - 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 with the most commonly used systems if most of the interaction is with a principal system.
- 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.

40

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.

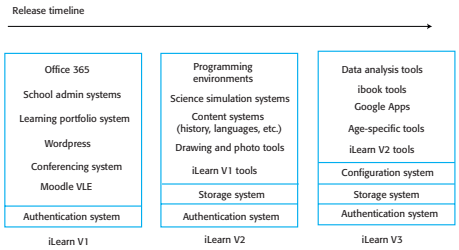
41

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. These tools might include archives for history, simulation systems for science, and programming environments for computing.
- Stage 3 adds features for user configuration and the ability for users to add new systems. Different versions of the system may be created for different age groups, with specialized or alternative tools, etc.

42

iLearn releases



43

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 an 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.

44

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.

45

Systems of systems architecture

46

General principles for architecting SoS

- Design systems so that they can deliver value if they are incomplete. There should be several “stable intermediate forms” so that a partial system can still do useful things.
- Be realistic about what can be controlled. Although the best performance may be achieved when an individual or group exerts control over the overall system and its constituents, overcontrolling the SoS is likely to lead to resistance from the individual system owners.
- Focus on the system interfaces. UIs should allow the system elements to interoperate and should not be too restrictive so that these system elements can evolve and continue to be useful participants in the SoS.

47

General principles for architecting SoS

- Provide collaboration incentives to constituent system owners: financial (pay per use or reduced operational costs), access (mutual sharing of data), or community (you get a say in community decision making).
- Design an SoS as node and web architecture:
  - Nodes are sociotechnical systems that include data, software, hardware, infrastructure (technical components), and organizational policies, people, processes, and training (sociotechnical).
  - The web is not just the communications infrastructure between nodes, but it also provides a mechanism for informal and formal social communications between the people managing and running the systems at each node.

48

General principles for architecting SoS

- Specify behavior as services exchanged between nodes. The development of service-oriented architectures now provides a standard mechanism for system operability. If a system does not already provide a service interface, then this interface should be implemented as part of the SoS development process.
- Understand and manage system vulnerabilities. In any SoS, there will be unexpected failures and undesirable behavior. It is critically important to try to understand vulnerabilities and design the system to be resilient to such failures.

49

Architectural frameworks

- Architectural frameworks such as MODAF and 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.

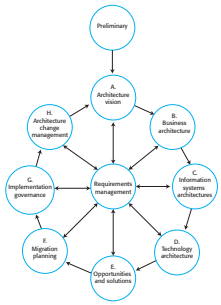
50

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.

51

TOGAF – Architecture Development Method



52

Architectural model management

- All architectural frameworks involve the production and management of a large set of architectural models. However, there are two issues to consider:
  - 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 an SoS.

53

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.
- As with all architectural patterns, real systems are usually based on more than one of these patterns.
- Architectural patterns can be effective in illustrating an SoS organization, without the need for detailed domain knowledge.

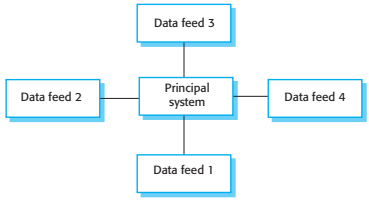
54

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.
- For example, to license a vehicle, you need to have both valid insurance and a roadworthiness certificate. When you interact with the vehicle licensing system, it itself interacts with two other systems to check that these documents are valid, one for insurances (run by insurance companies) and one for MOT (managed by the testing agencies licensed by the government to check the vehicles).

55

Systems as data feeds



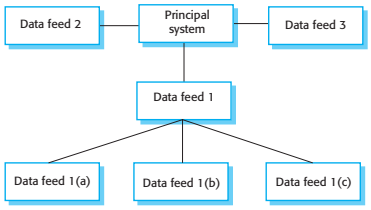
56

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.

57

Systems as data feeds with unifying interface



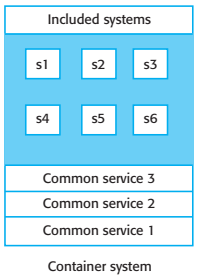
58

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.

59

Container systems



Container system

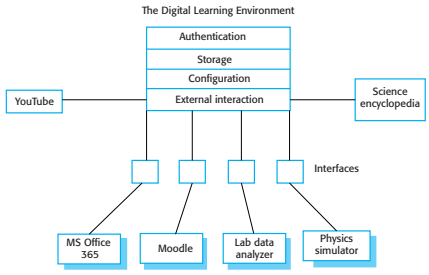
60

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.
- Other functionality comes from choosing existing systems such as a newspaper archive or a virtual learning environment and integrating these into the container.

61

iLearn as a container



62

ILearn container for Physics

- The previous figure shows a version of iLearn for Physics.
- As well as an office productivity system (Office 365) and a VLE (Moodle), this system includes simulation and data analysis systems.
- Other systems — YouTube and a science encyclopedia — are also part of this system.
  - However, these are not “approved” and so no container interface is available.
  - Users must log on to these systems separately and organize their own data transfers.

63

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.

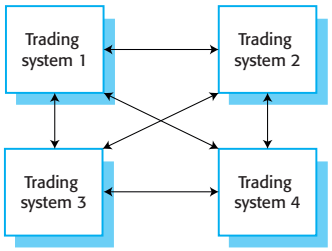
64

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.

65

Trading systems



66

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.

67

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.

68

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.

69

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.

70