Master of Science in Computer Engineering

Software Systems Engineering
a.a. 2021-2022
Gigliola Vaglini

1

Course objective

- To conduct a professional software project development that means taking into account the following aspects
  - Project requirements specification
  - Cost/Time evaluation
  - Planning
  - Design and implementation
  - Verification and validation of the product
- And to use suitable tools to perform each step

2

2

SW Engineering is a profession

- In the anglosaxon part of the world "software engineers" are known professionals.
  - Moreover, in the USA in 2010 there were about 900.000 "sw engineers" and more than the half of all engineers are sofware engineers.
- *Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm. To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession. Then a code of ethics is needed.*

3

3

Ethical dilemmas of an employee

- ✧ Disagreement in principle with the policies of senior management.
- ✧ Participation in the development of military weapons systems or nuclear systems.
- ✧ But also
- ✧ Release of a safety-critical system without finishing the testing of the system.

4

4

Issues of professional responsibility

- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Competence
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.

5

5

Issues of professional responsibility

- Intellectual property rights
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- Computer misuse
  - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

6

6

ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

7

7

The ACM/IEEE Code of Ethics

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

8

8

Ethical principles

1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

9

9

Examination

- 50% of the marks
  - Software Project
  - Project discussion
- 50% of the marks
  - Oral examination

10

10

Communication

- gigliola.vaglini@unipi.it
- mario.cimino@unipi.it

11

11

References

- Ian Sommerville: “Software Engineering”, Pearson ed., 10th edition
- Slides in the folder Class materials on the channel of the course
- http://www.computer.org/web/swebok/v3

12

12

Introduction

Lecture 1

13

13

From art to industry

- *The term "software engineering" was defined in a NATO conference in Garmisch, Oct. 1968*

14

14

Software engineering: why

- ✧The economies of ALL developed nations are dependent on software.
- ✧More and more systems are software controlled
- ✧Expenditure on software represents a significant fraction of the *Gross national product* (GNP) in all developed countries, included Italy.

15

15

Software engineering: what

- Software engineering is concerned with the practicalities of developing and delivering useful software.
  - Useful software should have an acceptable cost, then a *quantifiable* approach should apply to the development, operation, and maintenance of software.
  - So also organizational and financial constraints must be taken into account.

16

16

Software cost

- Consider that software costs exceed hardware costs: it exists a law to evaluate the development cost of a system: it is quadratic with the software dimension [Berra-Meo 2001]
- A high portion of software process cost is incurred in producing process and product documentation

17

17

Process documentation

- Effective management requires the process being managed to be visible. Then we need:
  - Plans, estimates and schedules produced by managers to predict and to control the software process.
  - Reports which report how resources were used during the process of development.
  - Standards which set out how the process is to be implemented.
  - Working papers that record the ideas and thoughts of the engineers working on the project and the rationale for design decisions.
  - E-mail messages, wikis, etc. These record the details of everyday communications between managers and development engineers.

18

18

Product documentation

- Product documentation includes
  - user documentation, which tells users how to use the software product, and
  - system documentation, which is principally intended for maintenance engineers.
    - Due to the long life of the software, maintainance cost can reach 70% of the software total cost.

19

19

Different use of Documentation

1. providing information for management
2. telling how to use the system
3. acting as a communication medium for members of the development team
4. Acting as an information repository for maintenance engineers.

5. Documentation may be essential evidence to be presented for system certification.

20

20

Producing documents

- Software engineers are usually responsible for producing most of this documentation.
- The set of documents that you have to produce for any system depends on the contract with the client, the type of system and its expected lifetime.
- For large projects, it is usually the case that documentation starts being generated well before the development process begins.

21

21

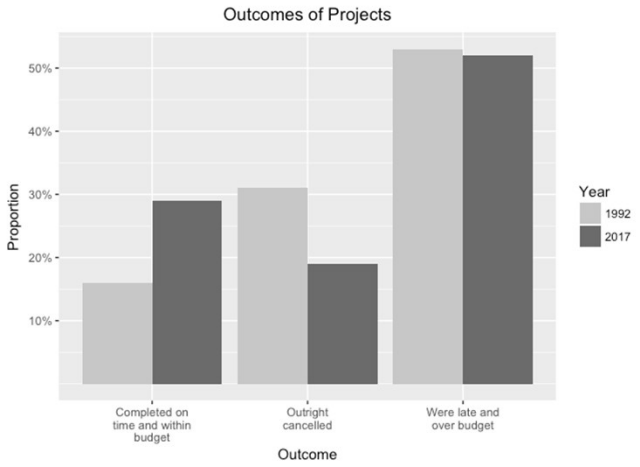
Other issues affecting software cost

- ✧ Heterogeneity
  - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- ✧ Business and social change
  - Business and society are changing incredibly quickly as new technologies become available. Software needs to be able to change.
- ✧ Security and trust
  - As software is intertwined with all aspects of our lives, it is essential that we can trust it.
- Quality control
  - More than 30% of software total costs are testing costs.

22

22

Productivity of ICT



23

23

Productivity of ICT

- Standish Report

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

- Failed : canceled projects or not useful results
- Changhelled: Out of time/cost, missing functionalities
- Successful: In time and in budget, with a satisfactory result

A 24

24

CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	100%	100%	100%

25

25

But essentially

- F. Brooks focused the main point :
  - "Complexity is an essential and non incidental property to software"
  - Moreover "the main part of such complexity is an arbitrary complexity
- Grady Booch, a developer of UML, has furtherly clarified the point :
  - "an essential complexity is not necessarily unmanageable, but it is simply not avoidable."

26

26

Complexity factors

- *Complex problems*
  - Competing requirings
  - Functional but also non functional requirements (security for example)
- *Complex products*
  - User friendliness is expensive
- *Complex development process*
  - Development Teams composed by many persons

27

27

More deeper on complexity

- An airplane is an artifact composed with about 100.000 elements, but the software on board in
  - The F-22 Raptor, the current U.S. Air Force frontline jet fighter, consists of about 1.7 million lines of code;
  - The F-35 Joint Strike Fighter consists of about 5.7 million lines of code;
  - The Boeing's new 787 Dreamliner consists of about 6.5 million lines of code
  - But a premium-class automobile "probably contains close to 100 million lines of software code"

28

28

Rapid software development

- Rapid development and delivery is often the most important requirement for software systems, for example, since
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
- Thus software for business systems
  - has to evolve quickly to reflect changing business needs. Development phases are inter-leaved and the system is developed as a series of versions with stakeholders involved in version evaluation

29

29

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

30

30

Agile manifesto

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

31

31

The principles of agile methods

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

32

32

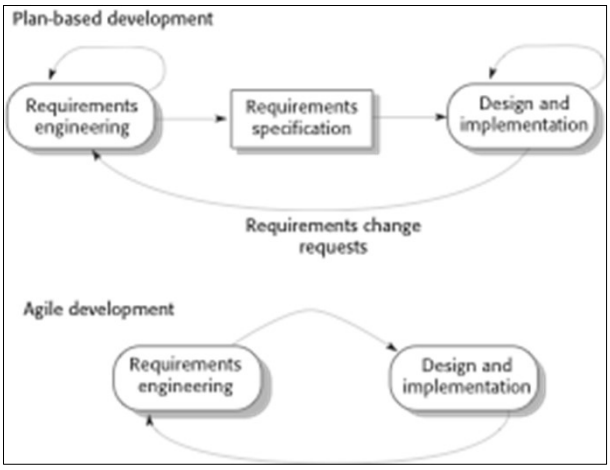
Plan-driven and agile development

- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model, incremental development is possible
  - Iteration occurs within activities.
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

33

33

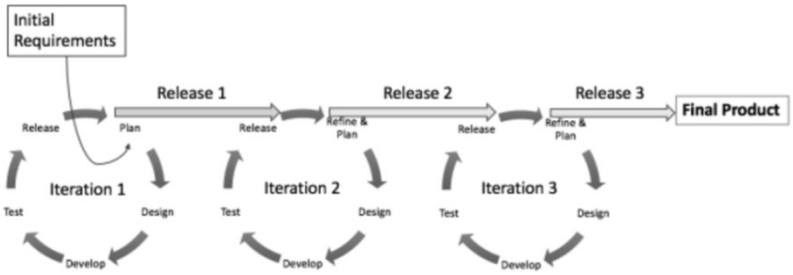
Plan-driven and agile specification



34

34

Does using an agile approach help overcome complexity and why would that be?



35

35

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	60%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

It can by failing earlier and restarting faster.

36

36

Agile method applicability

- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.

37

37

Problems with agile methods

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterises agile methods. Moreover, since the focus is on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

38

38

Agile methods and software maintenance

- Most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development above all when the original development team cannot be maintained.
- Thus the question is:
  - Are systems that are developed using an agile approach maintainable, given the emphasis of minimizing formal documentation?

39

39

Technical, human, organizational issues

- Most projects include elements of plan-driven and agile processes. Deciding on the balance depends on:
  - Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.
  - Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.
  - How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

40

40

Technical, human, organizational issues (2)

- What type of system is being developed?
  - Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).
- What is the expected system lifetime?
  - Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.
- What technologies are available to support system development?
  - Agile methods rely on good tools to keep track of an evolving design
- How is the development team organized?
  - If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.

41

41

Technical, human, organizational issues (3)

- How good are the designers and programmers in the development team?
  - It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code
- Is the system subject to external regulation?
  - If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.

42

42

In any case

- Both in plan-driven and agile approaches Project failures can occur and such occurrence is a cost too
  - what are the possible causes of failure?

43

43

Failure causes: legends

- A generic idea suffices to initiate, the details will be fixed successively
  - Bad specification causes the largest set of failed projects.
- If the project is late, it is possible to enlarge the manpower and respect the delivery date.
  - Brooks' Law: "Adding manpower to a late software project makes it later"

44

44

legends

- Changing requirements is not a problem, since software is flexible
  - Changing requirements is expensive, and changing requirements in the implementation phase is much more expensive than changing requirements in the design phase.
- When the software is in use the maintenance can be made easily error by error
  - The actual maintenance cost can reach 70% of the project cost.
- Past failures are not needed to be analyzed
  - Some recent exceptions

45

45

Good project principles

- Standard processes (and tools) allow (but do not assure) a good quality software

46

46

Assuring the quality of the software development process

- Internal organization
- Internal communication
- Intermediate products and milestones
- Results that can be reproduced

47

Assuring the quality of the software products

- Quality models
  - Quality characteristics are defined
    - Quality characteristics, functional and not functional, define what is the correct product that is expected
    - Correct software satisfies specification
  - Products are evaluated
    - Verification methods are used and proofs are established during the development process.
    - Software metrics are defined to evaluate the final product

48

New deals in SE: SE and the web

- The Web is now a platform for running applications, and organizations are increasingly developing web-based systems rather than local systems.
- The web has led to the availability of software services and the possibility of developing highly distributed service-based systems.
- Web services allow application functionalities to be accessed over the web.
- Cloud computing is an approach to the provision of services where applications run remotely on the 'cloud'.

49

49

Software as a service

- Software as a service (SaaS) involves hosting the software remotely and providing access to it over the Internet.
  - The software is owned and managed by a software provider, and not by the organizations using the software.
  - Users may pay for the software according to the amount of use they make of it or through an annual or monthly subscription.

50

50

Web software engineering

- Software reuse is the dominant approach for constructing web-based systems.
  - When building these systems, you think about how you can assemble them from pre-existing software components and systems.
- Web-based systems should be developed and delivered incrementally.
  - It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.
- User interfaces are constrained by the capabilities of web browsers.
  - Technologies such as AJAX allow rich interfaces to be created within a web browser.

51

51

Characteristics of software designed to be used as a service

- *Configurability* each organization has specific requirements
- *Multi-tenancy* each user of the software has the impression to be working with his/her copy of the system
- *Scalability* the system easily scales to accommodate an unpredictably large number of users

52

52

Services should be configurable for the needs of a specific organization

- Users from each organization are presented with an interface that reflects their own organization, this has also called Branding.
- Each organization defines its own rules that govern the use of the service and its data: it is possible to define specific Business rules and workflows.
- Service customers create individual accounts for their staff and define the resources and functions that are accessible to each of their users: Access control rules typical of an organization can be defined.

53

53

Multi-tenancy

- The system must be designed so that there is an absolute separation between the system functionalities and the system data.

54

54

Scalability

- Each system component is implemented as a simple stateless service that may be run on any server.
- The system uses asynchronous interaction so that the application does not have to wait for the result of an interaction (such as a read request).
- The system resources, such as network and database connections, are managed as a pool so that no single server is likely to run out of resources.
- The database uses fine-grain locking. That is, whole records are not locked out in the database when only part of a record is in use.

55

55

SaaS and SOA

- Service-oriented architecture is an approach to structuring a software system as a set of separate, stateless services.
- These last may be provided by multiple providers and may be distributed.
- Typically, transactions are short transactions where a service is called, does something then returns a result.

56

56

Web software engineering

✧The fundamental principles and ideas of software engineering apply to web-based software in the same way that they apply to other types of software system.

57

57

Distributed System Engineering

Lecture 2

58

58

✧ Distributed systems

- ✧ "… a collection of independent computers connected through a network that appears to the user as a single coherent system."
- ✧ From the point of view of a software engineer, information processing is distributed over several computers rather than confined to a single machine.

59

59

Complexity again

- ✧ Distributed systems are more complex than systems that run on a single processor.
  - ✧ Complexity arises because different parts of the system are independently managed.
  - ✧ There is no single authority in charge of the system so top-down control is impossible.

60

60

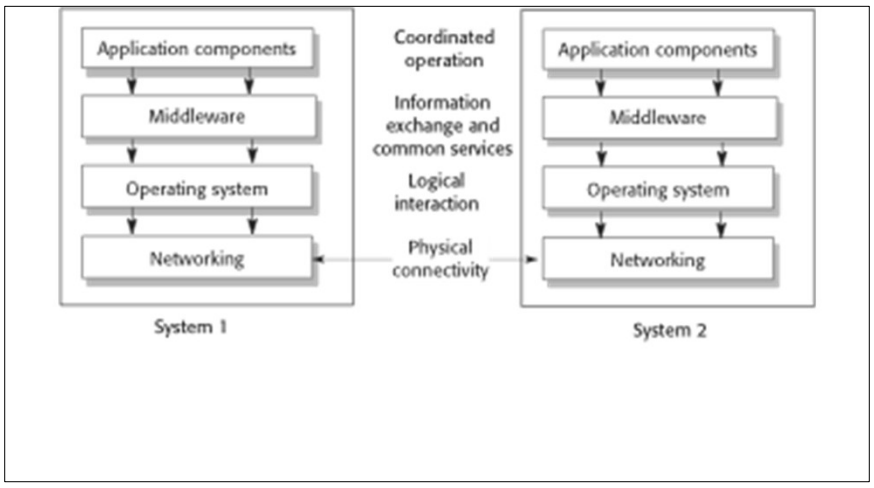
Middleware

- The components in a distributed system may be implemented in different programming languages and may execute on completely different types of processor. Models of data, information representation and protocols for communication may all be different.
- Middleware is the software that can manage these diverse parts, and ensure that they can communicate and exchange data.

61

61

Middleware in a distributed system



62

Middleware support

- The middleware coordinates interactions between different components in the system following different models
  - Procedural interaction, where one computer calls on a known service offered by another computer and waits for a response.
  - Message-based interaction, involves a computer sending information to another computer. There is no necessity to wait for a response.
  - The middleware provides location transparency in that it isn't necessary for components to know the physical locations of other components.
- The middleware provides implementation of services that may be required by several components
  - By using these common services, components can easily inter-operate and provide user services in a consistent way.

63

63

Distribute system and Internet: Client-server computing

- The user interacts with a program running on the local computer (e.g. a web browser or phone-based application). This interacts with another program running on a remote computer (e.g. a web server).
- The remote computer provides services, such as access to web pages, which are available to external clients.

64

64

Service-oriented Architecture

65

65

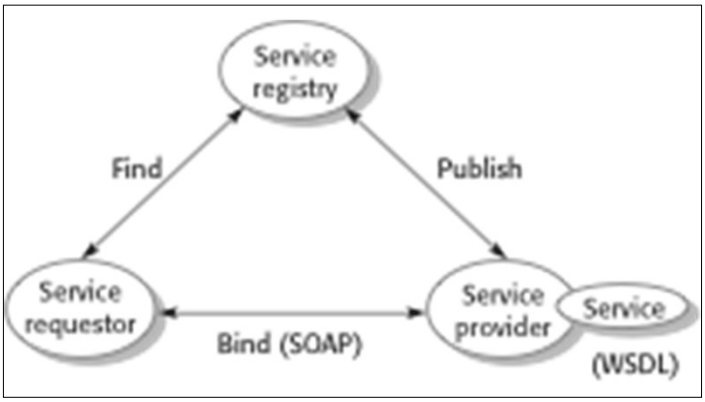
Service-oriented architectures

- A means of developing distributed systems where the components are stand-alone services
- Services may execute on different computers
- Standard protocols have been developed to support service communication and information exchange

66

66

Service-oriented architecture



67

67

Standards

- SOAP
  - A message exchange standard that supports service communication
- WSDL (Web Service Definition Language)
  - This standard allows a service interface and its bindings to be defined
- WS-BPEL
  - A standard for workflow languages used to define service composition

68

68

Web service standards

69

69

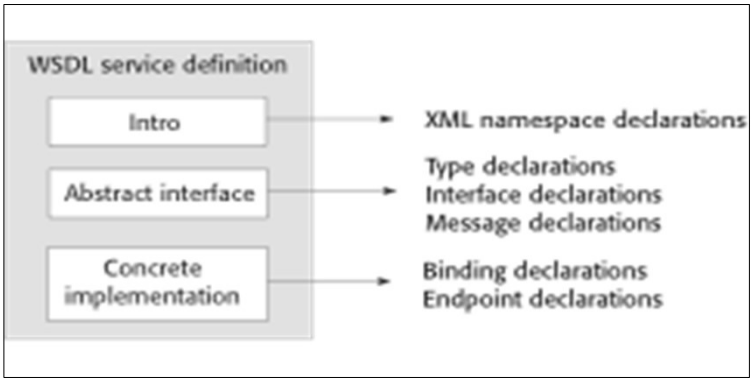
WSDL specification components (What-How-Where)

- The 'what' part of a WSDL document, called an interface, specifies what operations the service supports, and defines the format of the messages that are sent and received by the service.
- The 'how' part of a WSDL document, called a binding, maps the abstract interface to a concrete set of protocols. The binding specifies the technical details of how to communicate with a Web service.
- The 'where' part of a WSDL document describes the location of a specific Web service implementation.

70

70

Organization of a WSDL specification



71

71

Service-oriented software engineering

- The approaches to software engineering have to evolve to reflect the service-oriented approach to software development
  - Service engineering. The development of dependable, reusable services
    - Software development for reuse
  - Software development with services. The development of dependable software where services are the fundamental components
    - Software development with reuse

72

72

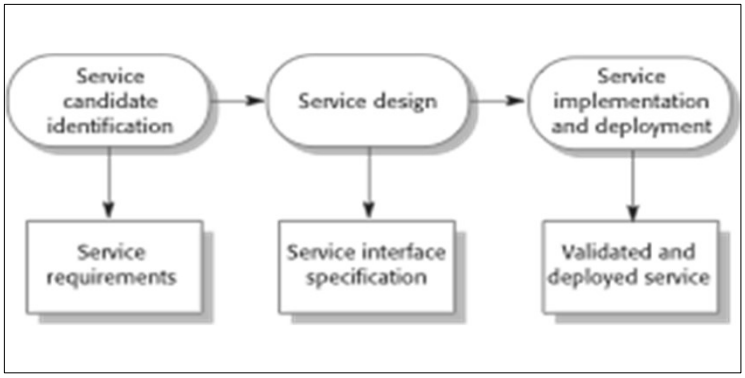
Task and entity-oriented services

- Task-oriented services are those associated with some activity.
- Entity-oriented services are like objects. They are associated with a business entity such as a job application form.

73

73

Realizing a new service: the service engineering process



74

74

Stages of service engineering

- Service candidate identification, where you identify possible services that might be implemented and define the service requirements.
- Service design, where you design the logical and WSDL service interfaces.
- Service implementation and deployment, where you implement and test the service and make it available for use.

75

75

Service candidate identification

- Three fundamental types of service
  - Utility services that implement general functionality used by different business processes.
  - Business services that are associated with a specific business function e.g., in a university, student registration.
  - Coordination services that support composite processes such as ordering.

76

76

Service classification: remind

Utility or business services may be entity- or task-oriented, coordination services are always task-oriented.

	Utility	Business	Coordination
Task	Currency converter Employee locator	Validate claim form Check credit rating	Process expense claim Pay external supplier
Entity	Document style checker Web form to XML converter	Expenses form Student application form	

77

77

An example

- A large company, which sells computer equipment, has arranged special prices for approved configurations for some customers.
- To facilitate automated ordering, the company wishes to produce a catalog service that will allow customers to see their dedicated prices.
- Goods are ordered through the web-based system of each company that accesses the catalog as a web service.
- The catalog is created by a supplier to show which good can be ordered

78

78

What the supplier wants

- Specific versions of catalog should be created for each client
- Catalog shall be downloadable
- The specification and prices of up to 6 items may be compared
- Browsing and searching facilities shall be provided
- A function shall be provided that allows the delivery date for ordered items to be predicted
- Virtual orders shall be supported which reserve the goods for 48 hours to allow a company order to be placed

79

79

Constraints

- ✧ Access shall be restricted to employees of accredited organisations
- ✧ Prices and configurations offered to each organisation shall be confidential
- ✧ The catalog shall be available from 0700 to 1100
- ✧ The catalog shall be able to process up to 10 requests per second

80

80

Service interface design: functionalities

Operation	Description
MakeCatalog	Creates a version of the catalog tailored for a specific customer. Includes an optional parameter to create a downloadable PDF version of the catalog.
Compare	Provides a comparison of up to six characteristics (e.g., price, dimensions, processor speed, etc.) of up to four catalog items.
Lookup	Displays all of the data associated with a specified catalog item.

81

81

Service interface design (2)

Operation	Description
Search	This operation takes a logical expression and searches the catalog according to that expression. It displays a list of all items that match the search expression.
CheckDelivery	Returns the predicted delivery date for an item if ordered that day.
PlaceOrder	Reserves the number of items to be ordered by a customer and provides item information for the customer's own procurement system.

82

82

Logical interface design

Operation	Inputs	Outputs	Exceptions
MakeCatalog	*mcIn* Company id PDF-flag	*mcOut* URL of the catalog for that company	*mcFault* Invalid company id
Compare	*compIn* Company id Entry attribute (up to 6) Catalog number (up to 4)	*compOut* URL of page showing comparison table	*compFault* Invalid company id Invalid catalog number Unknown attribute
Lookup	*lookIn* Company id Catalog number	*lookOut* URL of page with the item information	*lookFault* Invalid company id Invalid catalog number

83

83

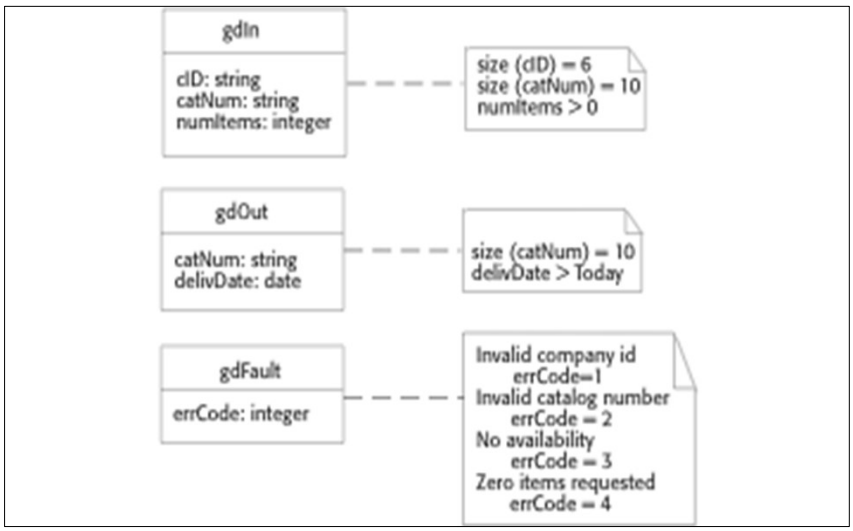
Logical interface design

Operation	Inputs	Outputs	Exceptions
Search	*searchIn* Company id Search string	*searchOut* URL of web page with search results	*searchFault* Invalid company id Badly formed search string
CheckDelivery	*gdIn* Company id Catalog number Number of items required	*gdOut* Catalog number Expected delivery date	*gdFault* Invalid company id Invalid catalog number No availability Zero items requested
PlaceOrder	*poIn* Company id Number of items required Catalog number	*poOut* Catalog number Number of items required Predicted delivery date Unit price estimate Total price estimate	*poFault* Invalid company id Invalid catalog number Zero items requested

84

84

Message structure design: an example. UML definition of input and output messages for CheckDelivery



85

85

Last step

- WSDL description
  - The logical design is converted to a WSDL description

86

86

Service implementation and deployment

- Programming services using a standard programming language or a workflow language
- Services then have to be tested by creating input messages and checking that the output messages produced are as expected
- Deployment involves publicing the service and installing it on a web server. Current servers provide support for service installation

87

87

Software development with services

- Existing services can be composed and configured to create new composite services and applications
- The basis for service composition is often a workflow
  - Workflows are logical sequences of activities that, together, model a coherent business process

88

88

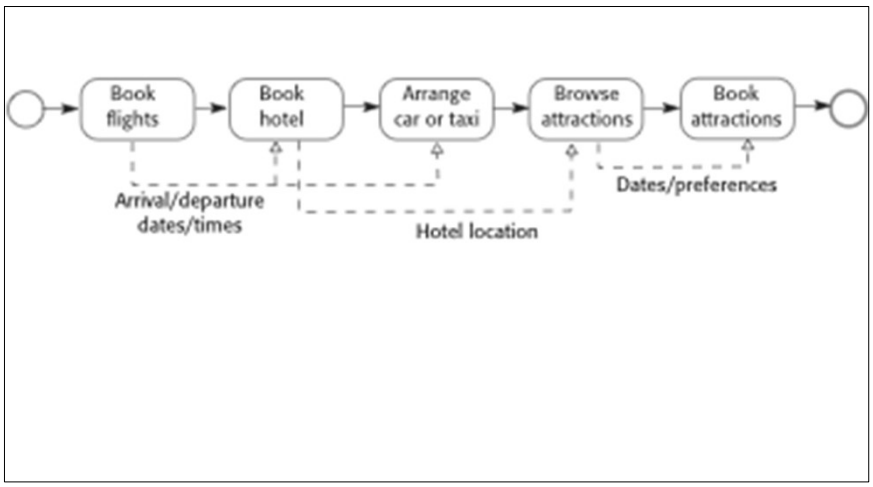
Example

- For example, provide a travel reservation services which allows flights, car hire and hotel bookings to be coordinated
- A workflow is the following

89

89

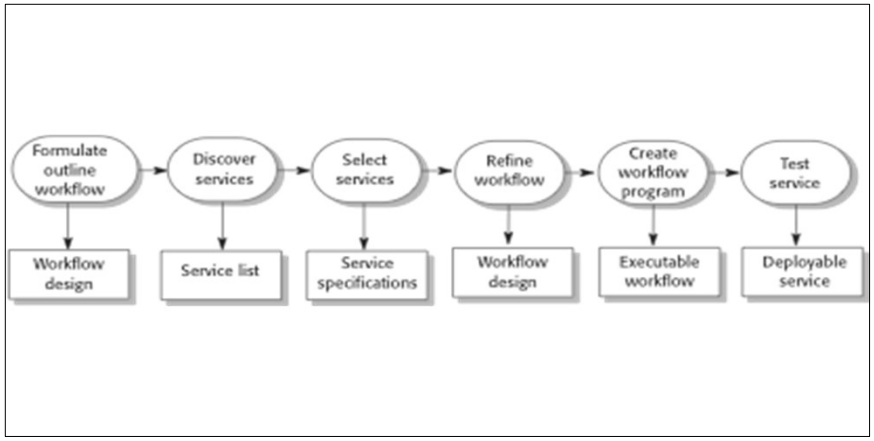
Vacation package workflow



90

90

Service construction by composition: general workflow of activities



91

91

Construction by composition

- *Formulate outline workflow*
  - In this initial stage of service design, you use the requirements for the composite service as a basis for creating an 'ideal' service design.
- *Discover services*
  - During this stage of the process, you search service registries or catalogs to discover what services exist, who provides these services and the details of the service provision.
- *Select possible services*
  - Your selection criteria will obviously include the functionality of the services offered. They may also include the cost of the services and the quality of service (responsiveness, availability, etc.) offered.

92

92

Construction by composition

- *Refine workflow*.
  - This involves adding details to the abstract description and perhaps adding or removing workflow activities.
- *Create workflow program*
  - During this stage, the abstract workflow design is transformed to an executable program and the service interface is defined. You can use a conventional programming language, such as Java or a workflow language, such as WS-BPEL.
- *Test completed service or application*
  - The process of testing the completed, composite service is more complex than component testing

93

93

Workflow design and implementation

- WS-BPEL (Web Services Business Process Execution Language) is an XML-standard for workflow specification.
- Unfortunately, WS-BPEL descriptions are long and unreadable; graphical workflow notations, such as BPMN, are more readable and WS-BPEL can be generated from them.
- In inter-organisational systems, separate workflows are created for each organisation and linked through message exchange: in this way we have interacting workflows.

94

94

RESTful web services

- Current web services standards have been criticized as 'heavyweight' standards that are over-general and inefficient.
- REST (REpresentational State Transfer) is an architectural style based on transferring representations of resources from a server to a client through a global identifier (URL).
- This style underlies the web as a whole and is simpler than SOAP/WSDL for implementing web services.
- RESTFul services involve a lower overhead than so-called 'big web services' and are used by many organizations implementing service-based systems that do not rely on externally-provided services.

95

95

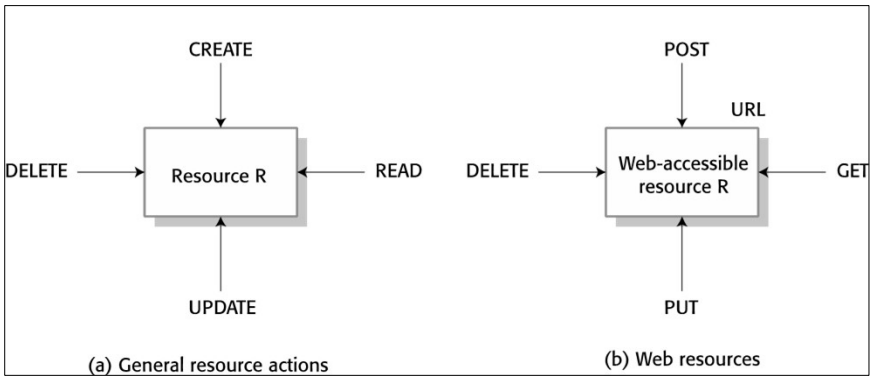
Resources

- The fundamental element in a RESTful architecture is a resource.
- Essentially, a resource is simply a data element such as a catalog or a document.
- In general, resources may have multiple representations i.e. they can exist in different formats.
  - MS WORD
  - PDF….

96

96

Operations



97

97

Operation functionality

- POST is used to create a resource. It has associated data that defines the resource.
- GET is used to read the value of a resource and return that to the requestor in the specified representation, such as XHTML, that can be rendered in a web browser.
- PUT is used to update the value of a resource.
- DELETE is used to delete the resource.

98

98

Resource access

- When a RESTful approach is used, the data is exposed and is accessed using its URL.
- Therefore, the weather data for each place in the database, might be accessed using URLs such as:
  - http://weather-info-example.net/temperatures/edinburgh
- Invokes the GET operation and returns a list of maximum and minimum temperatures.

99

99

Query results

- The response to a GET request in a RESTful service may include URLs.
- If the response to a request is a set of resources, then the URL of each of these may be included.
  - http://weather-info-example.net/temperatures/edinburgh-scotland
    http://weather-info-example.net/temperatures/edinburgh-australia
    http://weather-info-example.net/temperatures/edinburgh-maryland

100

100

Disadvantages of RESTful approach

- When a service has a complex interface and is not a simple resource, it can be difficult to design a set of RESTful services to represent this.
- There are no standards for RESTful interface description so service users must rely on informal documentation to understand the interface.

101

101

Service testing

- ✧ Testing is intended to find defects and demonstrate that a system meets its functional and non-functional requirements.
- ✧ Service testing is difficult as (external) services are 'black-boxes'. Testing techniques that rely on the program source code cannot be used.

102

102

Main service testing problems

- External services may be modified by the service provider thus invalidating previously completed tests
- The non-functional behaviour of the service is unpredictable because it depends on load.
- If services have to be paid for as used, testing a service may be expensive.
- External services may rely on the failure of other services which cannot be simulated.

103