Systems Development

Lecture 7: OOA&D in Practice

Contents

- Software requirements
- Construction and iteration
- Contingency theory
- Strategy with OOA&D

Contents

- Software requirements
  - Unclear requirements
  - Empirical study of practice
- Construction and iteration
- Contingency theory
- Strategy with OOA&D

Unclear Requirements

- There are many examples of problems with requirements
- Unclear requirements have been denoted as the main source of problems in system development projects
- Example: Amanda (1993-2008)
  - The IT system for Arbejdsformidlingen (the national centre for unemployed) ended up costing 650+ million kroner (original budget 214 million kroner)

  - It did not fit with the work of the employees; for example, they had to go through 40-50 screens to register a single person as unemployed
  - The time spent per registration went from 10-20 minutes to 1 hour
- Do you know of similar examples and the causes?
- Is there any way we can avoid such problems?

Empirical Study of Practice

- Lauesen, S. and Vinter, O. (2001) Preventing Requirement Defects: An Experiment in Process Improvement. *Requirements Eng*. 6:37–50.
- Requirement defect (definition):
  - The product works as intended by the programmers, but doesn't match the surroundings (Context: AD+PD)
- Examples:
  - Users and customers are not satisfied with the product. They find it too difficult to use, unable to support certain user tasks, etc.
  - The program doesn't cooperate properly with existing, surrounding software

Experiment

- Find cost-effective ways to avoid requirement defects in the products. Research approach:
  - Analyse the defects in present products, identify techniques that could prevent them, and try the best of them in new projects.
- The experiment was conducted at Brüel & Kjaer (B&K) in Denmark
- B&K manufacture professional equipment for sound and vibration measurement, and more than half of the product developers are software people

- B&K develop products according to a waterfall model where phases can overlap to some extent
- They talk about phases such as requirements specification, design, programming, module test, integration test, etc.
- From the integration test on, B&K routinely records all defects detected by programmers, in-house product testers, marketing and customers.

Requirements to the System: Examples

- A good way of assuring the measurement quality is to examine the measured spectra. This allows the experienced user to determine the quality of the measurement:
- (R-25) During the measurement the application must show the latest measured spectrum.
- . . .
- It is sometimes impossible to measure some of the desired points. It may be too hot in the environment, or there may not be enough space to position the probe:
- (R-35) The application must be able to display all results, even if some of the points have not been measured.
- Requirement spec: A total of 20 pages with 107 requirements

Analysis of Defect Reports

- About 800 defect reports a few months after product release
- 107 reports were analysed in detail
- Major sources of defects:
  - Missing requirements, i.e., requirements that had not been written down in the spec and were not otherwise transferred to developers (45 cases): ignored or forgotten (21 of the 45 or not recognized although they had been present in the domain all the time (24 of the 45)
  - Mistaken tacit requirements, i.e., the developer somehow knew about the demand, but made a wrong solution (24 cases): made a wrong guess (nine of the 24) or couldn't resolve apparently conflicting or inconsistent demands (15 of the 24)
  - Mistaken specs, i.e., written requirements that were implemented incorrectly (14 cases): a simple mistake (four of the 14), a misunderstanding of the spec (two of the 14), inconsistent requirements (two of the 14) or broad requirements that were not fully implemented everywhere, e.g. that 'the interface shall follow the Windows style guide' (six of the 14).
  - Defects relating to external software: misunderstood how external software worked or the external software didn't work correctly or didn't fulfil expectations
- They also classified each defect according to the quality factor (McCall) that was impaired, e.g., functionality, usability or maintainability. Almost 70% of the defects related to usability (ease of understanding and use)

Defects and Potential Techniques

- Defects in existing product:
  - 60% of the defects related to unstated demands (tacit requirements)
  - Almost 70% of the defects had to do with ease of understanding or ease of use (usability)
  - Most related to the user interface and misunderstood interfaces to third-party software
- Potential techniques:
  - Identified 44 techniques from literature or practice
  - For each defect they identified the techniques that might find or prevent the defect – and with what probability.
  - About 10 techniques were worth considering in a project of this kind

Measured Effect in a New Project

- A new project:
  - The user tasks were studied directly and the user interface was designed and usability tested before any part of it was programmed.
- Obstacles:
  1. Some top techniques were useful in one kind of project, but much less important in other projects
  2. The organisational surroundings may block the use of some techniques
  3. Developers have difficulties using many new techniques at the same time
  4. Unforeseen events, such as a new project manager, can overturn earlier decisions to use a certain technique.

- In the new project (experiences):
  1. The number of usability problems per screen picture was reduced by about 70% (they expected18%)
  2. The project was the first one ever in the company that had been completed on time and without stress
  3. The product sold twice as many units as comparable products and at twice the unit price

The Empirical Study

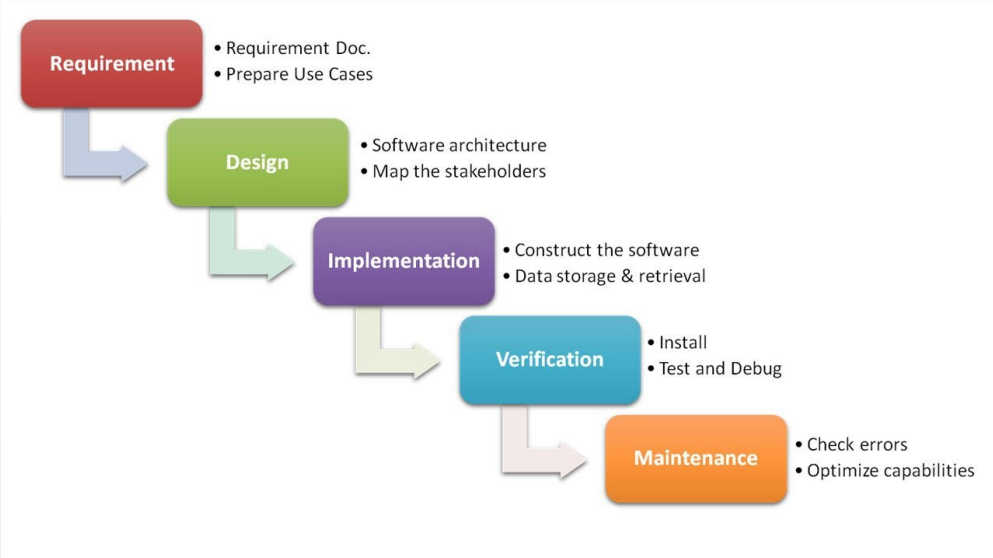
- Very interesting results that emphasize the importance of clear requirements
- What are the strengths and weaknesses of this study?

Contents

- Software requirements
- Construction and iteration
  - Waterfall model
  - Iterative model
  - Examples
- Contingency theory
- Strategy with OOA&D

Waterfall

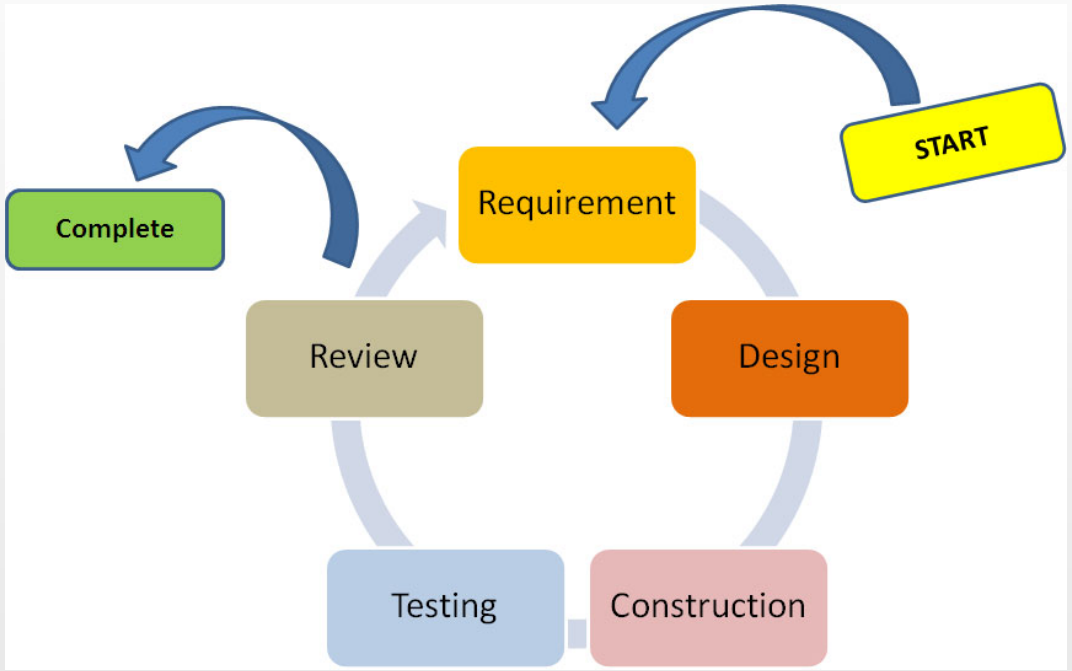
- Construction
- Example: 8 queen problem
- The problem can be specified precisely and in detail
- Development through a number of phases (refinement)
- Each phase has a clear purpose



- **Challenges:**
  - Based only on specifications but they are difficult to produce and understand
  - Difficult to get the users to describe their work
  - Non-technical aspects are difficult to specify
  - Requirements are changing over time
  - Works only when we know exactly what we want and we are able to describe it precisely and unambigously
  - Fedback loops become necessary
  - Many negative effects of the systems developed

Iterative

- Evolution with prototypes
- Example: support to nurses in patient care
- No set of clear requirements to depart from
- The understanding of the problem is changing during development
- Development through a series of cycles
- The requirements and the system are improved in each iteration



Examples

- Give an example of a software development project and suggest to what extent waterfall and/or iteration would be relevant

Example 1

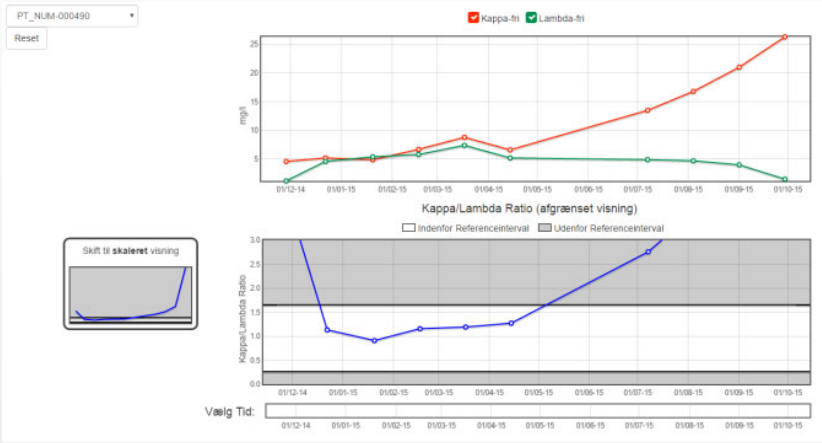
- The B&K systems were developed with a waterfall model
- Involved a number of phases from requirements to product release
- The new project involved prototype development and evaluation in the requirements phase (some iteration)

Example 2

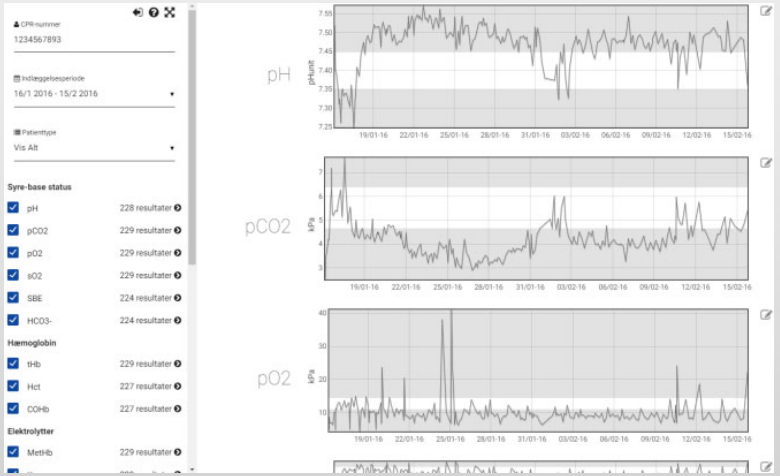
- Visualisation of blood test results for diagnosing
  - Adam Viggo Glistrup and Dennis Dalgaard (2016) UI Prototypes - Identification of software requirements in a highly complex domain. Master thesis, Aalborg University, Department of Computer Science.
- Impossible for developers to learn enough about the diagnosis process to enable them to produce a prototype
- The preferences of doctors
- Boundary object: physical object with different representation, drawing, illustrations, and more
- Used Boundary objects to facilitate discussion with the medical doctors and illustrate options

Different Prototypes

- Multiple myeloma



- Intensive care



Contents

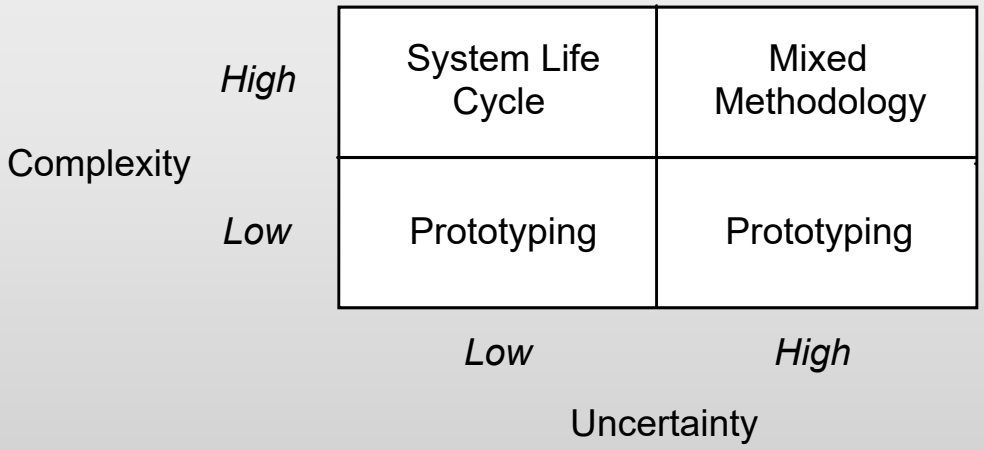
- Software requirements
- Construction and iteration
- Contingency theory
  - Davis
  - Burns and Dennis
  - Principle of limited reduction
- Strategy with OOA&D

Contingency Theory (1)

- How do you choose between waterfall/life cycle and prototyping?
- The relevance of these categories of methods can be determined from contingency factors
- Davis (1982):
- Analyze uncertainty in terms of the following four factors:
  - the organizational and technical context of the system
  - the future computer system
  - the experience and skills of the users
  - the experience and skills of the system developers
- Selection of approach:
  - If uncertainty is low: base requirements determination on an informal approach or on analysis of existing systems
  - If uncertainty is high: use specifications or prototypes

Contingency Theory (2)

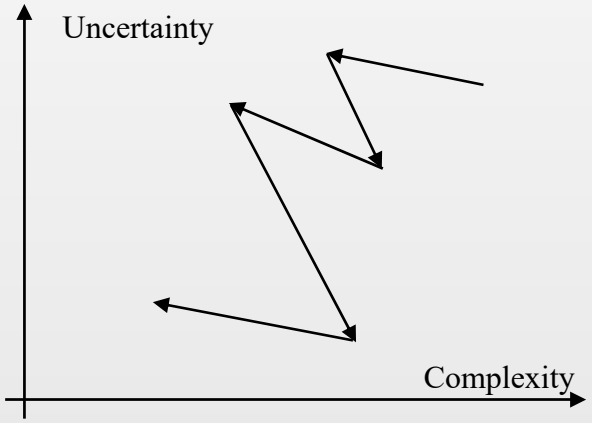
- Burns and Dennis (1985)
- Extend and refine Davis' idea by introducing a distinction between complexity and uncertainty
- Uncertainty:
  - the degree of structuredness that characterizes the users' work
  - the degree of understanding the users have about their work
  - the degree of experience and training of the system developers
- Complexity:
  - project size
  - number of users
  - volume of new information
  - complexity of new information



Principle of Limited Reduction

- Mathiassen and Stage (1992)
- Contingencies should be analysed dynamically throughout a development project

Quantity	**Too much**	**Too little**
Quality	**Too difficult**	**Too unreliable**
	Complexity	Uncertainty



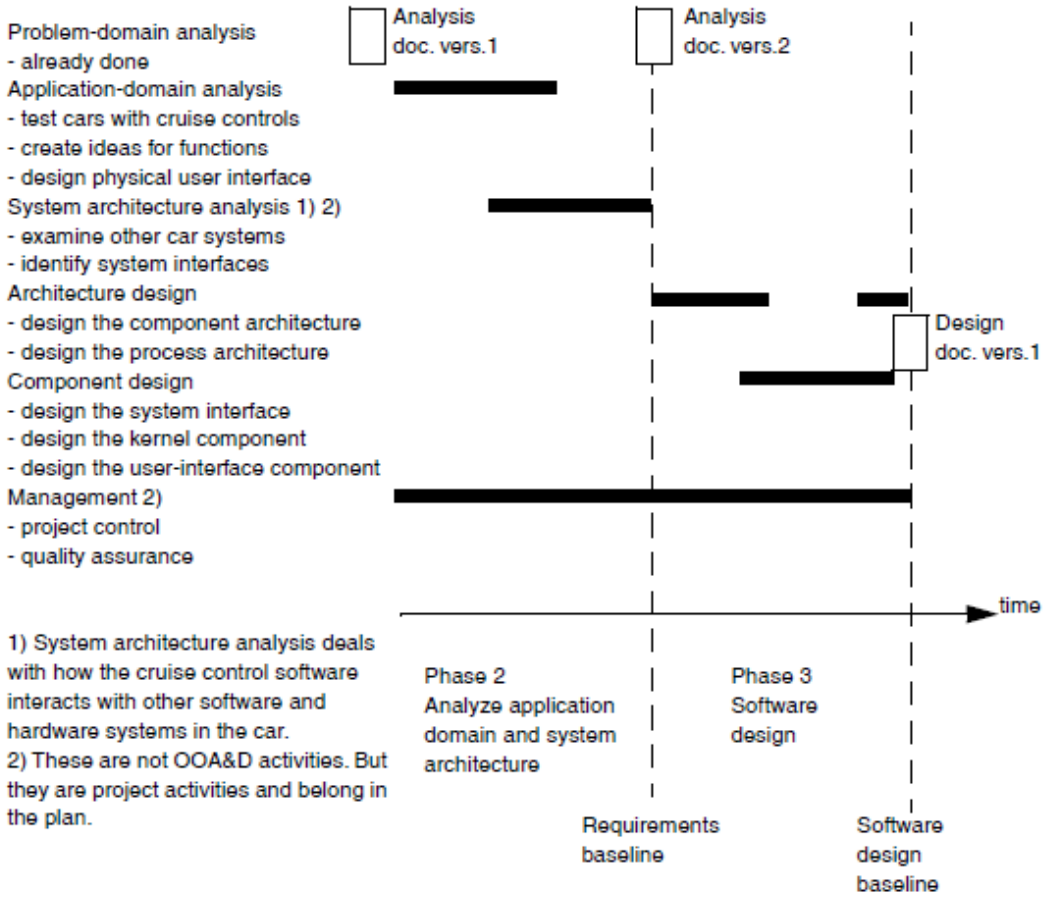
- Principle of limited reduction:
  - Relying on an analytical mode of operation *to reduce complexity* introduces new sources of uncertainty requiring experimental countermeasures.
  - Relying on an experimental mode of operation *to reduce uncertainty* introduces new sources of complexity requiring analytical countermeasures.

Contents

- Software requirements
- Construction and iteration
- Contingency theory
- Strategy with OOA&D
  - Result
  - Key concepts
  - Activities

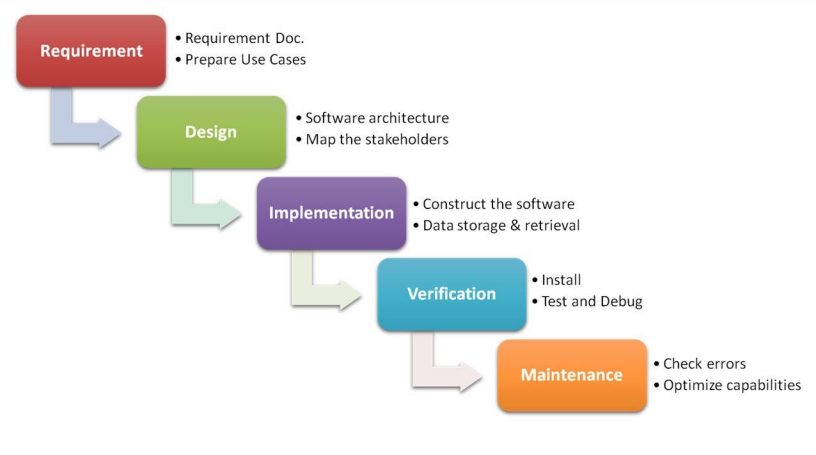
Strategy: Result

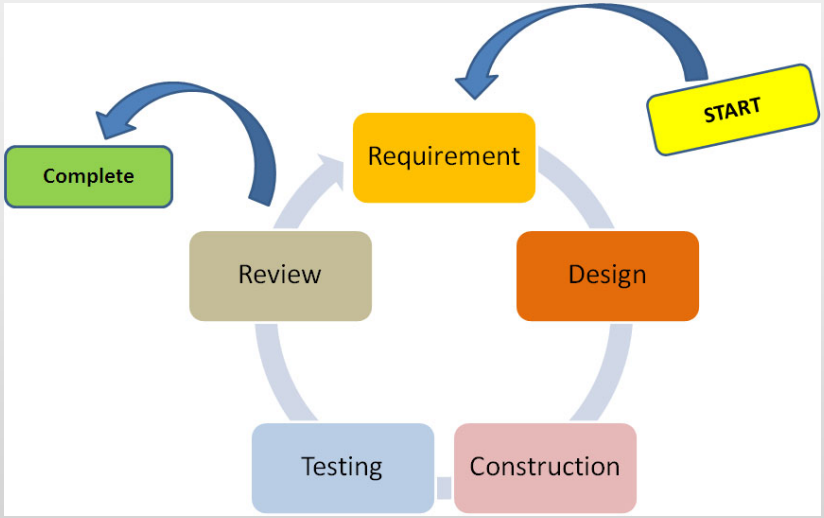
Strategy: Focus on architecture design, where the greatest difficulties are. But first complete analysis of application domain and system architecture to establish the requirements baseline.



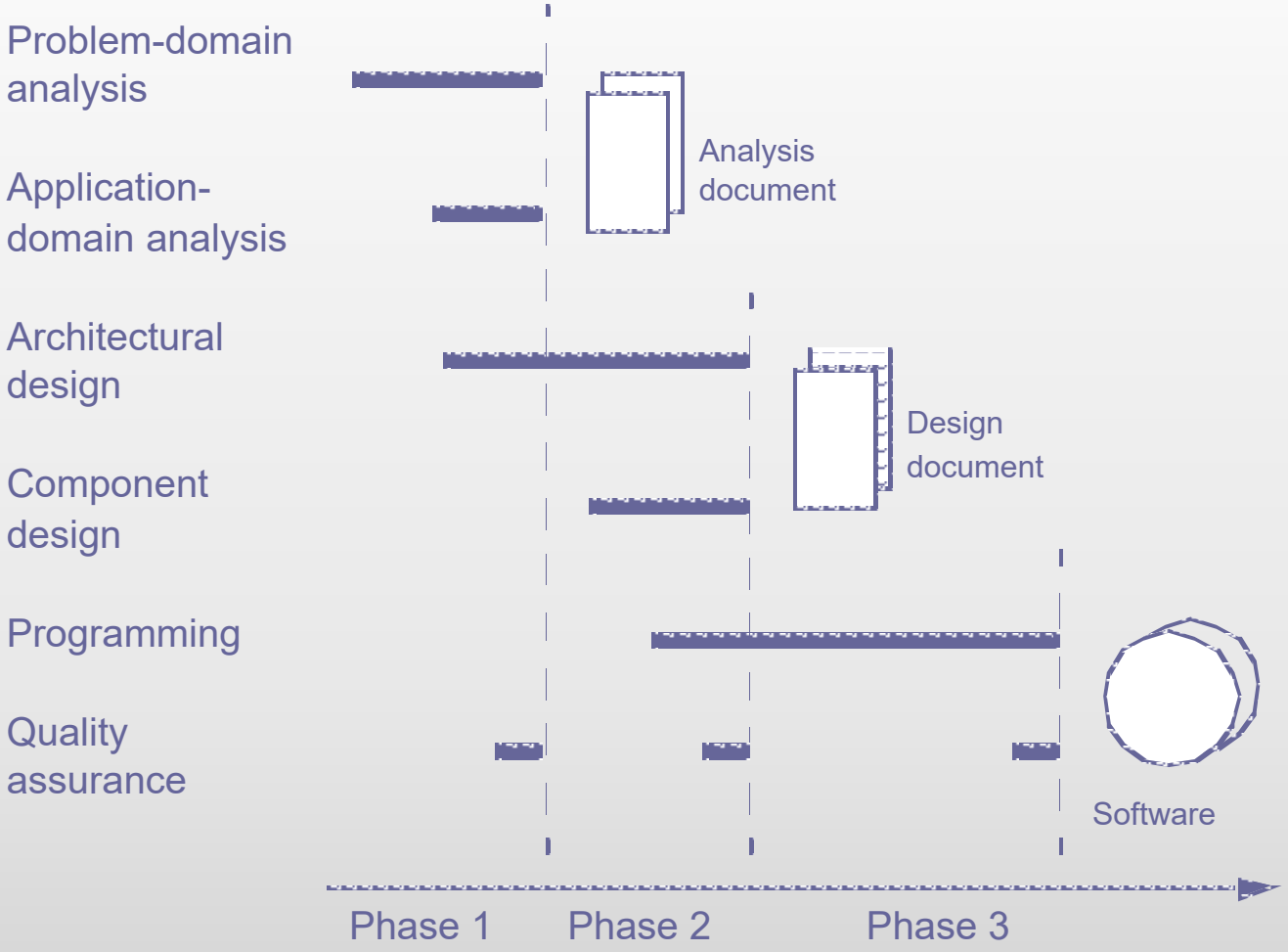
Strategy: Development Approach

- Which development approach is compatible with OOA&D
  - Waterfall model
  - Iterative model
- Both ...

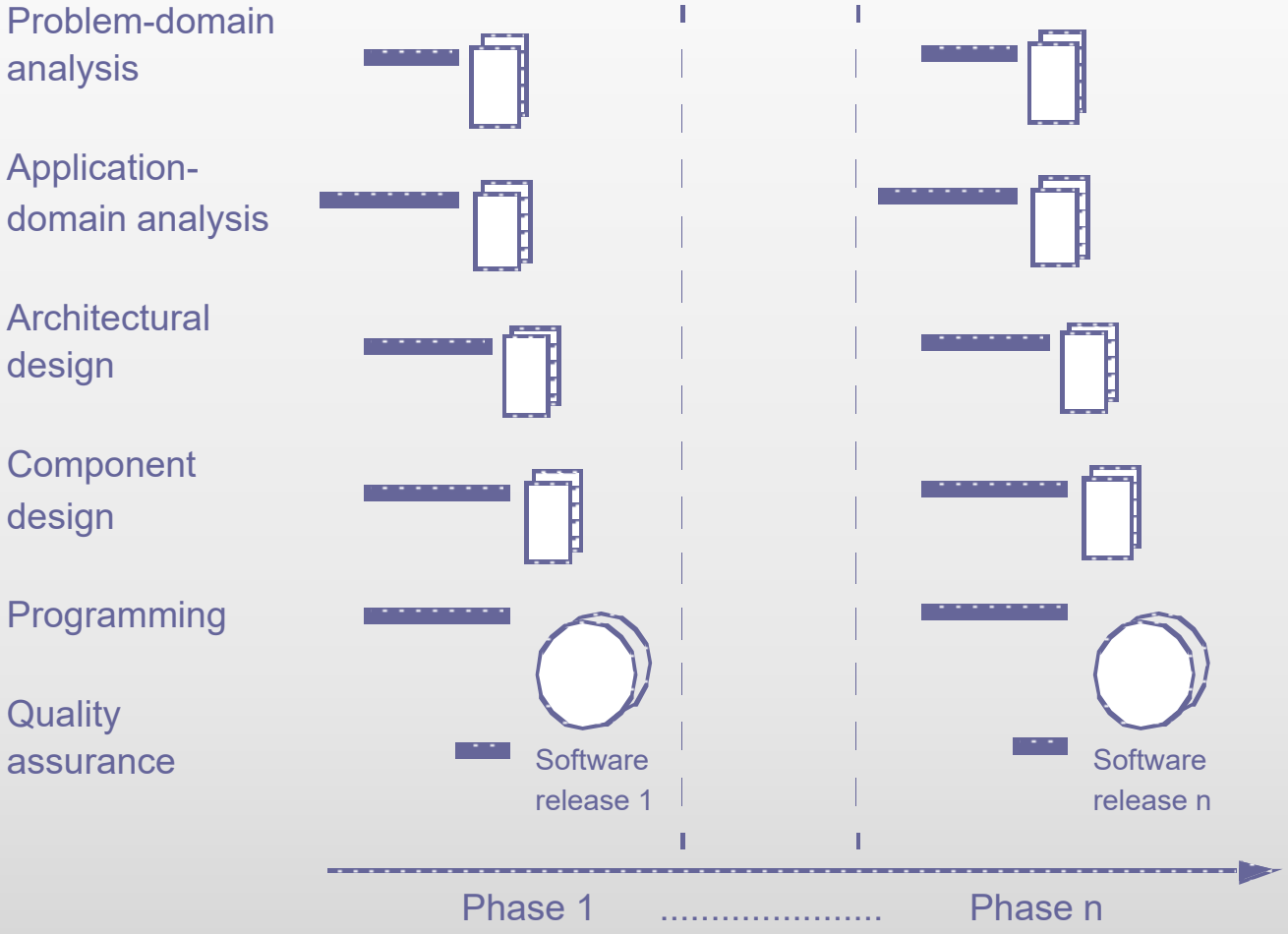




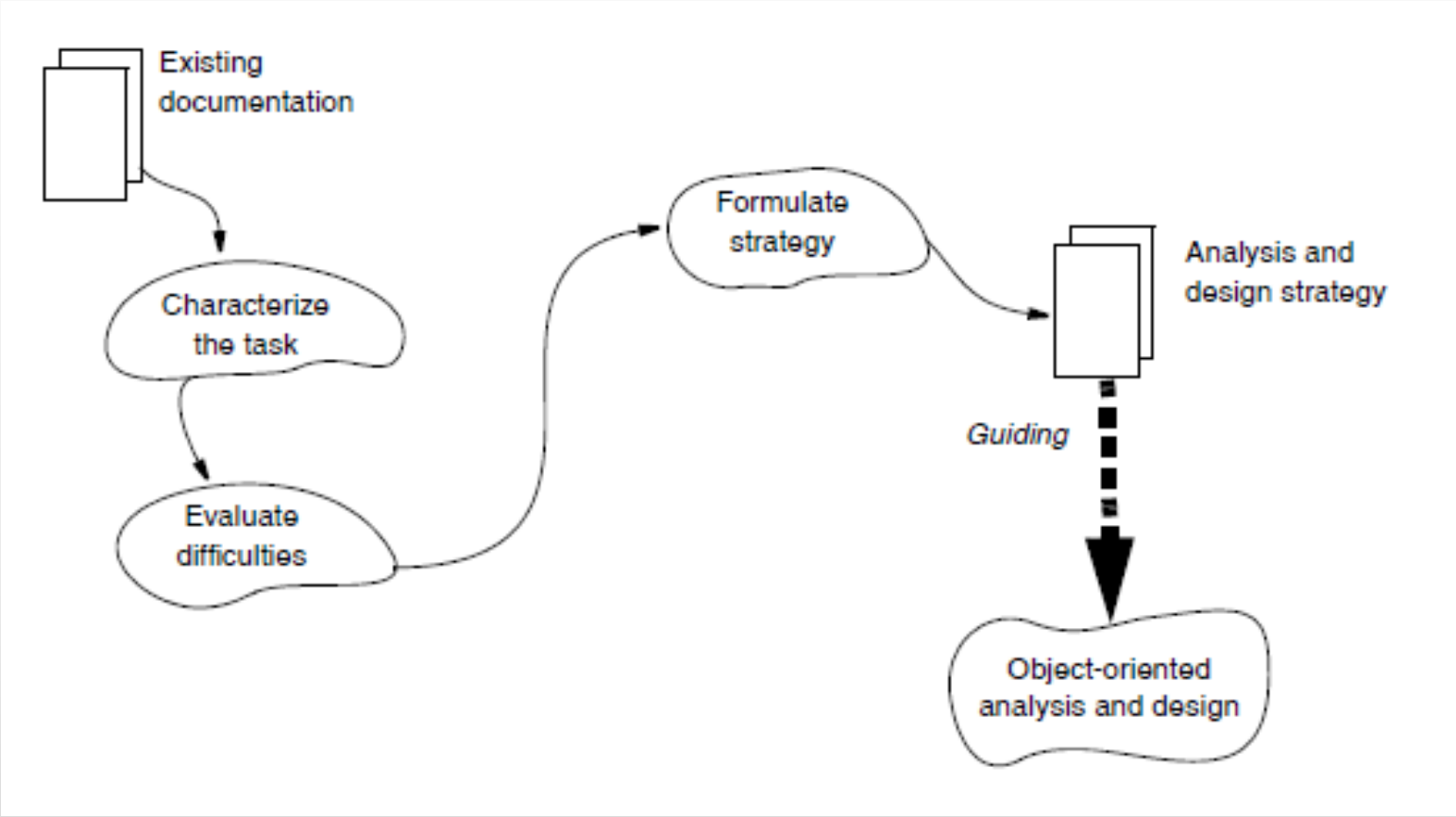
Traditional, top-down



Use-case driven, architecture-centric, and incremental



Strategy: Activities



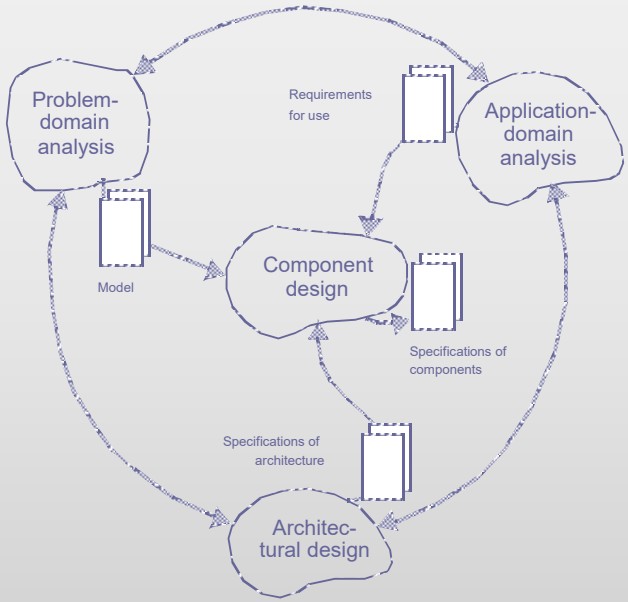
Characterize the Task

- Describe the basic characteristics of the task
- Embodies the fundamental idea of contingency theory
- Turn into numbers (quantify)
  yes = 0
  perhaps/don't know = 1
  no = 2

Problem domain:	
1. Has the level of ambition been agreed upon with the customer?	yes
2. Is the model of the object system clear and simple?	perhaps
3. Is the problem domain stable?	yes
Application domain:	
4. Are the requirements to the system's use complete, clear, and simple?	no
5. Do the users have experience with this type of computerized system?	yes
6. Is the system's functionality well known to the system developers?	yes
7. Is the application domain and user group clear and uniform?	yes
8. Are there existing procedures or traditions for using the system?	yes
9. Are the requirements to the system's response time and security manageable?	yes
10. Will it be possible to introduce the system without major changes?	no
Technical platform:	
11. Is the technical platform, including libraries, etc., tested, simple, and available?	no
12. Can the requirements be realized on the technical platform?	don't know
13. Can the technical platform be established independently of other projects?	no
14. Is the relation to other systems well defined and simple?	no
Project organization:	
15. Do the system developers have insight into the problem domain?	yes
16. Do the system developers have insight into the application domain?	yes
17. Are the system developers experienced on the technical platform?	no
18. Is there close contact with the users during development?	no
19. Are the system developers experienced in object-oriented development?	yes
Special aspects:	
20. Is there experience with similar systems?	no
21. Does the project look like previous development projects?	don't know
22. Are there only few special conditions?	no

Evaluate Difficulties

- Which of the overall activities of OOA&D appears to be most difficult
- Tailor the strategy to resolve the most significant difficulties first



Analysis of the problem domain:	
1. Has the level of ambition been agreed upon with the customer?	0
2. Is the model of the object system clear and simple?	1
3. Is the problem domain stable?	0
5. Do the users have experience with this type of computerized system?	0
15. Do the system developers have insight into the problem domain?	0
18. Is there close contact with the users during development?	2
19. Are the system developers experienced in object-oriented development?	0
20. Is there experience with similar systems?	2
21. Does the project look like previous development projects?	2
22. Are there only few special conditions?	2
Total difficulty in analysis of the problem domain:	**9/20**
Analysis of the application domain:	
1. Has the level of ambition been agreed upon with the customer?	0
4. Are the requirements to the system's use complete, clear, and simple?	2
5. Do the users have experience with this type of computerized system?	0
7. Is the application domain and user group clear and uniform?	0
8. Are there existing procedures or traditions for using the system?	0
9. Are the requirements to the system's response time and security manageable?	0
10. Will it be possible to introduce the system without major changes?	2
16. Do the system developers have insight into the application domain?	0
18. Is there close contact with the users during development?	2
20. Is there experience with similar systems?	2
21. Does the project look like previous development projects?	2
22. Are there only few special conditions?	2
Total difficulty in analysis of the application domain:	**12/24**
Design of the system's architecture:	
9. Are the requirements to the system's response time and security manageable?	0
10. Will it be possible to introduce the system without major changes?	2
11. Is the technical platform, including libraries, etc., tested, simple, and available?	2
12. Can the requirements be realized on the technical platform?	2
13. Can the technical platform be established independently of other projects?	2
14. Is the relation to other systems well defined and simple?	2
17. Are the system developers experienced on the technical platform?	2
19. Are the system developers experienced in object-oriented development?	0
20. Is there experience with similar systems?	2
21. Does the project look like previous development projects?	2
22. Are there only few special conditions?	2
Total difficulty in design of the architecture:	**18/22**

Strategy: Summary

Purpose	• To prioritize and order the analysis and design activities.
Concepts	• Requirements: A system's externally observable behavior. • Conditions: The technical, organizational, and human opportunities and limits involved in performing a task. • Strategy: A general approach for performing a task.
Principles	• Tailor the strategy. • Identify the difficult requirements and conditions. • Maintain degrees of freedom.
Results	• A strategy for the analysis and design task.