# A good practice is to be complete

## Preventing incompleteness problems in software architectures

**Frank van den Berk**

# Presentation setup



- **Part I: Incompleteness often leads to problems**
- **Part II: Ways to prevent incompleteness problems**

# Examples of architecture problems



- **Professional system I**
  - **Concurrency got too little attention in architecture**
  - **>6 months testing, lots of race conditions and performance problems**

- **Consumer system**
  - **Predictable performance enhancements not supported by architecture**
  - **Out of business now, because they could not keep up with competition**

# Examples of architecture problems



- **Professional system II**
  - **No adequate mechanisms for diagnostics provided by architecture**
  - **Field problems took ages to solve, angry customers, no time for new developments**

- **Professional system III**
  - **Exception behavior hardly analyzed by architects**
  - **Lots of refactoring in test phase, missed market window (twice)**

# Examples of architecture problems



- Professional system IV
  - No mechanisms defined to configure the software
  - Some desired configurations of the system could not be realized - they did not have the courage to try

- Setting up new architectures (several examples)
  - Architects tend to focus on a few promising and new aspects
  - Architecture is incomplete, so either project is cancelled or other aspects are solved in many different ways by many different engineers

# Incompleteness often leads to problems …



***Most software consists of two parts:***

- ***A part that was designed by the architects***
- ***A part that <u>should have been</u> designed by the architects***

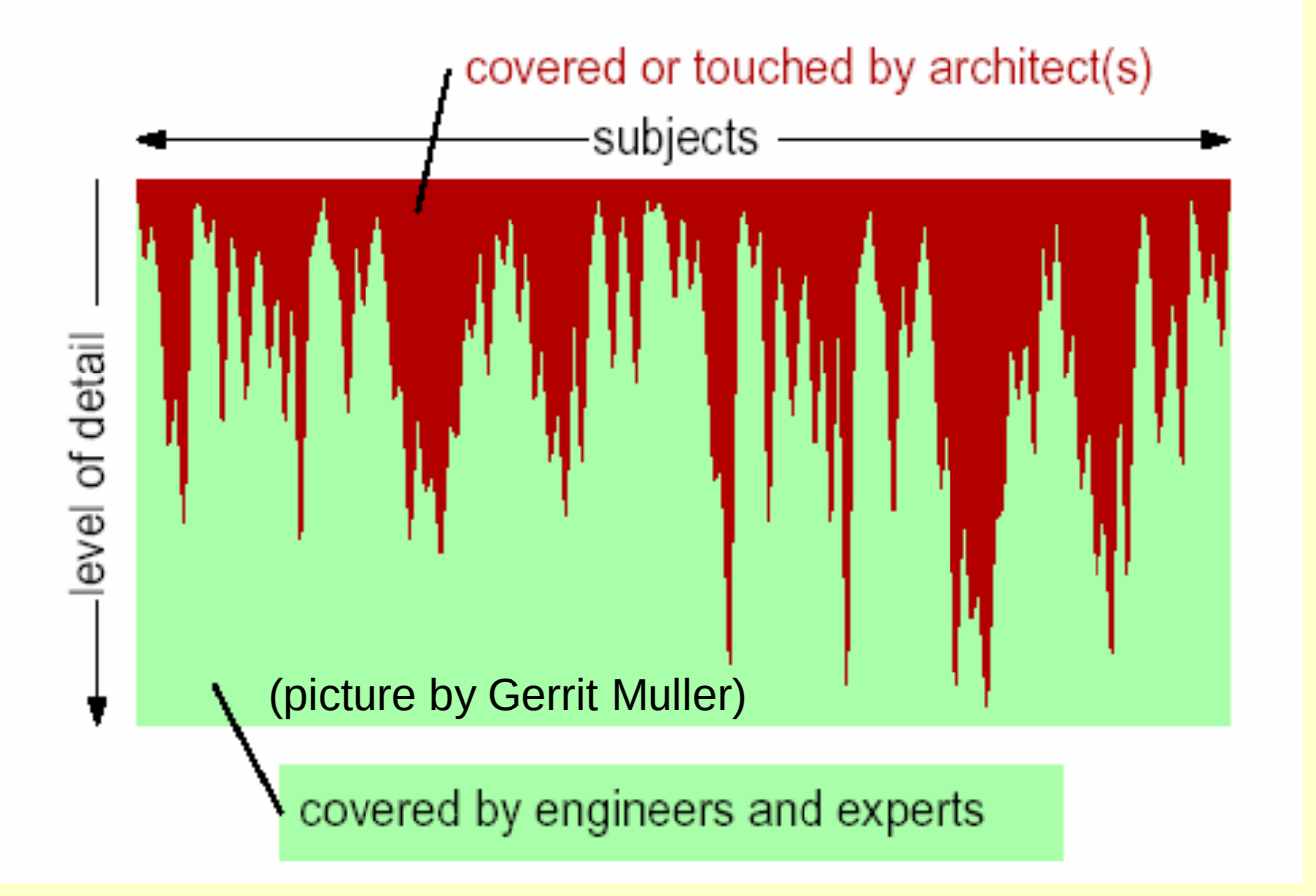
***The latter part often leads to the biggest problems..***

# Two “dimensions” of incompleteness



1. Important subjects not covered

2. Too little detail (gaps with engineers), or too much detail (lost time)



# A good practice is to be complete ..?



## There is much to gain from good practices to prevent incompleteness problems

**Some considerations:**

- **An architecture is never complete**
  - **… but there are many types of incompleteness; not all of them are harmless**
- **“Completeness” has a different meaning for every system**
  - **… but the practices to prevent incompleteness are very similar**

# Ways to prevent incompleteness problems



***Good practices to decide <u>what</u> has to be done:***

- **Regular "quick scans" on completeness**
- **Explicit reviews, involving the stakeholders**
- **Clear boundaries for the architect's tasks**

***Good practices to ensure that it <u>is</u> done:***

- **A plan**
- **A delegation structure**
- **Explicit risk management**

# Ways to prevent incompleteness problems



***Good practices to decide <u>what</u> has to be done:***

- **Regular "quick scans" on completeness**
- **Explicit reviews, involving the stakeholders**
- **Clear boundaries for the architect's tasks**

***Good practices to ensure that it <u>is</u> done:***

- **A plan**
- **A delegation structure**
- **Explicit risk management**

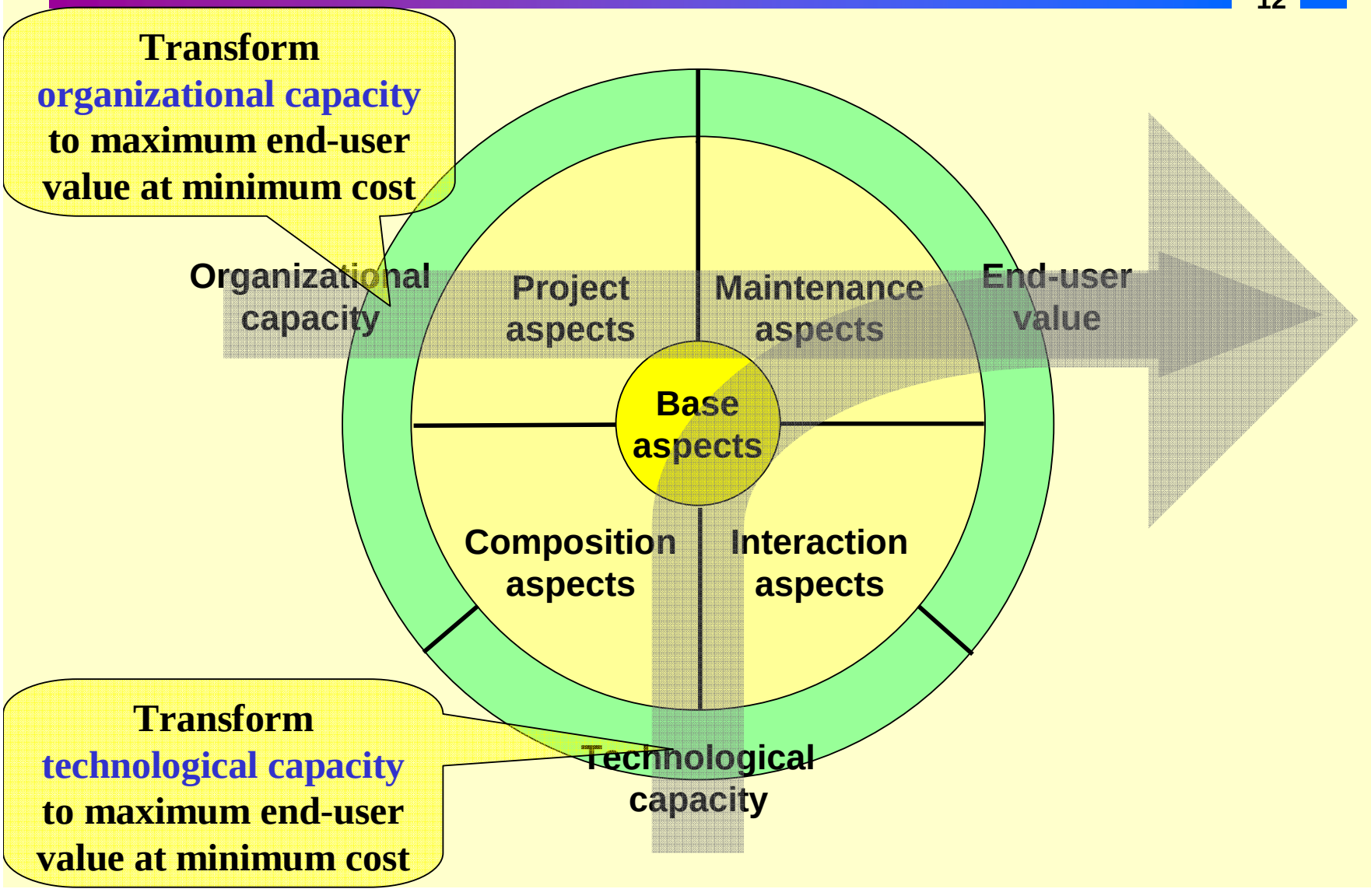
# Regular “quick-scan” on completeness



1. **Use a viewpoint-model (4+1, Soni, …), a checklist, a template, or combinations**
2. **Translate the abstractions from the model / checklist to *your* situation**
   - **Create your own domain-specific checklist**
3. **Complete the checklist *together with the stakeholders***
4. **Scan for missing parts in your architecture *together with the stakeholders***

# A simple but effective checklist

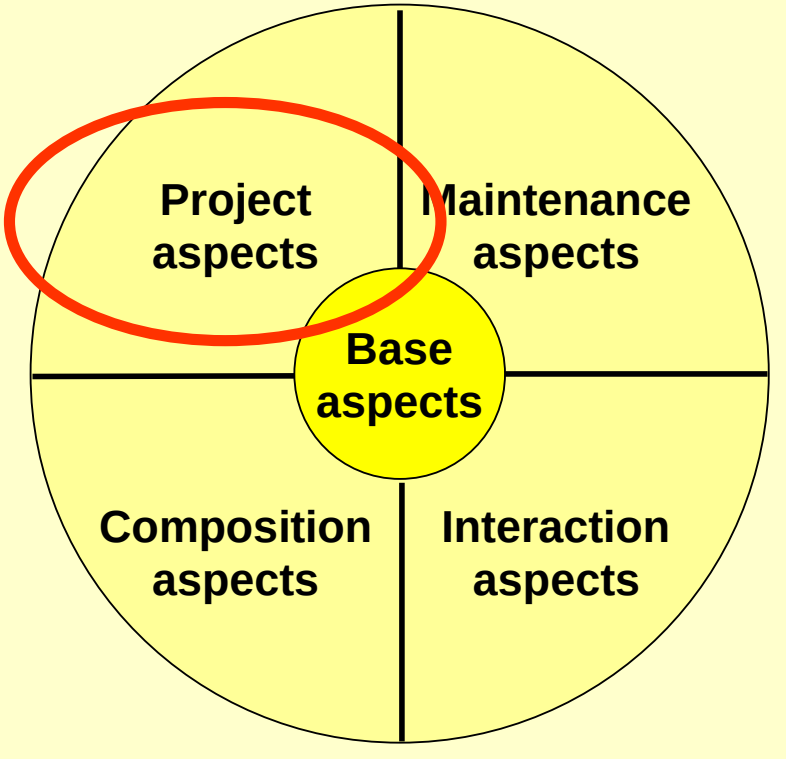




# A simple but effective checklist

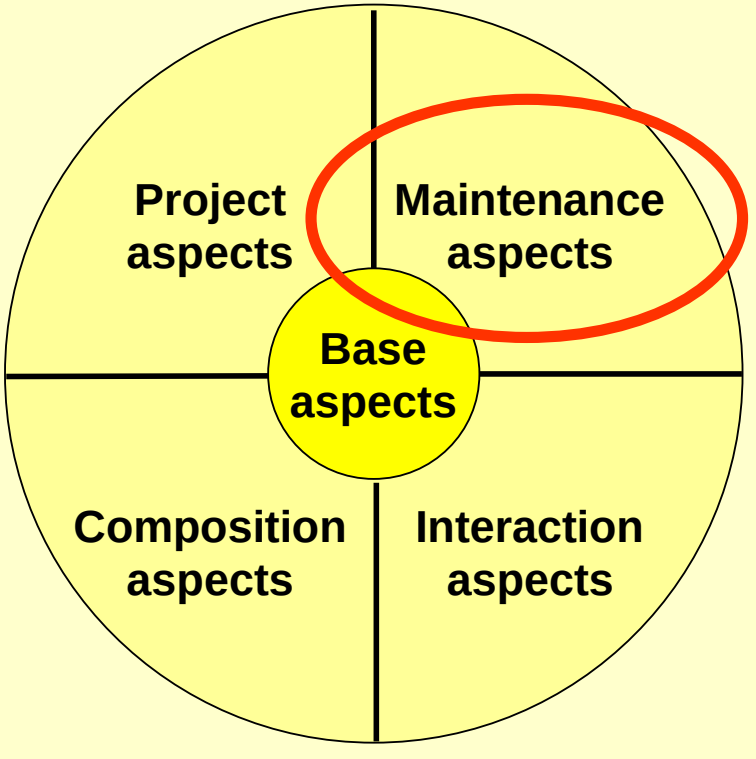


- Does everyone know what to expect when from the architects?
- Is the architecture definition planning synchronized with other relevant plans?
- Does the architecture support early mitigation of risks?
- Does the architecture fit the (distribution of) knowledge and skills in the organization?
- Did we plan sufficient architecture (deployment) verification activities?
- Are the integration dependencies between the software elements defined?
- Do we need additional work products, such as stubs and test drivers, to support the test and integration strategy of the project?



# A simple but effective checklist

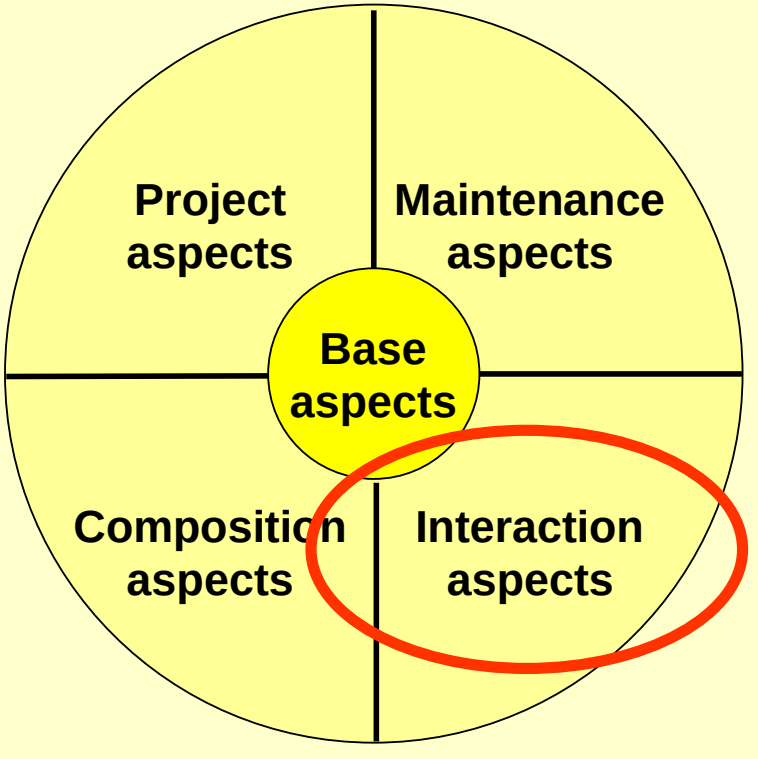





- Do we know how to (de-)install both first and new software versions?
- Do we know how to (de-)install software patches?
- Do we know how to get information on the local software-, and system configuration?
- Do we know how to diagnose the software and the system?
  - Its performance and other key aspects?
  - In case of problems?
- Do we know how to handle replacements of system parts, including new versions of system parts?
- Do we provide all necessary maintenance information?

# A simple but effective checklist



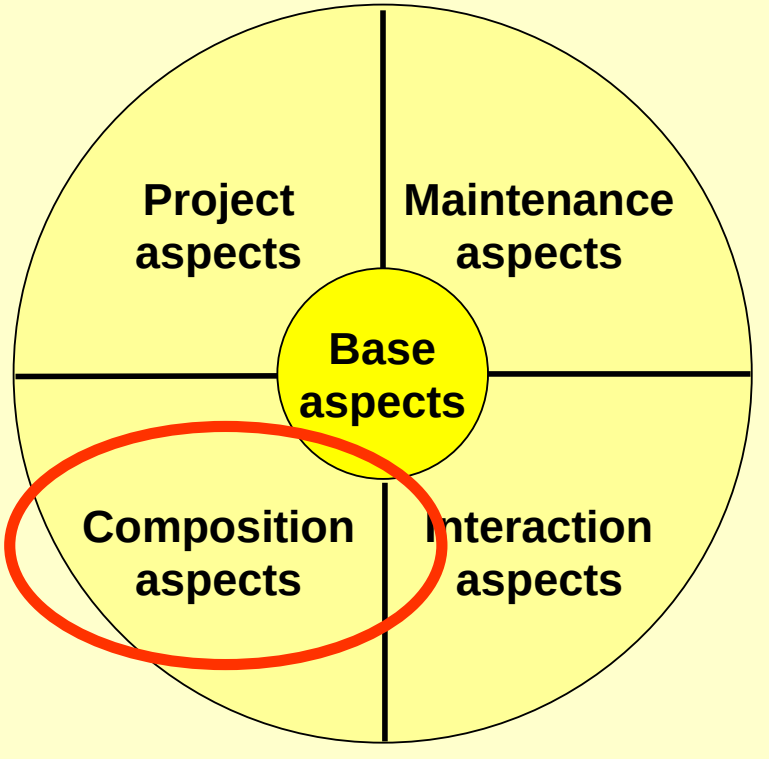



- Did we investigate important interaction scenarios with all relevant human users, hardware elements, and other software systems?
- Did we specify the interfaces with all relevant human users, hardware elements, and other software systems?
- Can we handle all possible (combinations of) interaction scenarios correctly and with sufficient performance?
- Are the available resources (memory, bandwidth, CPU power, …) sufficient to handle all possible (combinations of) interaction scenarios?
- Can we handle faults adequately?
- Are all *critical* interaction scenarios identified and under control?
- Do we know how to start up and shut down the system?

# A simple but effective checklist



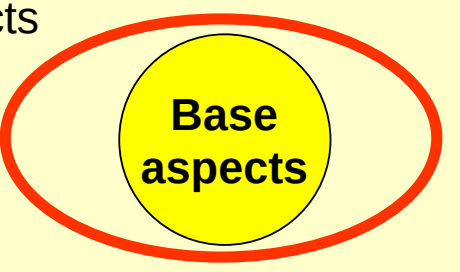
- Are all elements to implement the architecture defined?
  - Did we define the scope, the interfaces, the behavior, and the contribution to overall system qualities or budgets for each component?
- Are all *critical* components and interfaces identified and under control?
- Is the mapping of the software elements to the hardware defined? Does it fit?



# A simple but effective checklist



- Does every stakeholder know when and how the architects should be consulted?
- Does every stakeholder know how to get the latest relevant information on the architecture?
- Did we identify and explain the styles and patterns we use?
- Do we have a glossary with all important terms and definitions?
- Do we have rules and mechanisms for basic recurring aspects in the software?
  - Design-, coding-, and naming conventions
  - Communication and synchronization mechanisms
  - Configuration mechanisms
  - ...
- Do we have rules and guidelines on how to configure and use generic tools and resources, such as development environments, resource files, and software libraries?
- Do we know what we want to re-use, and do we have rules and guidelines for these assets?



# Ways to prevent incompleteness problems



***Good practices to decide <u>what</u> has to be done:***

- **Regular "quick scans" on completeness**
- **Explicit reviews, involving the stakeholders**
- **Clear boundaries for the architect's tasks**

***Good practices to ensure that it <u>is</u> done:***

- **A plan**
- **A delegation structure**
- **Explicit risk management**

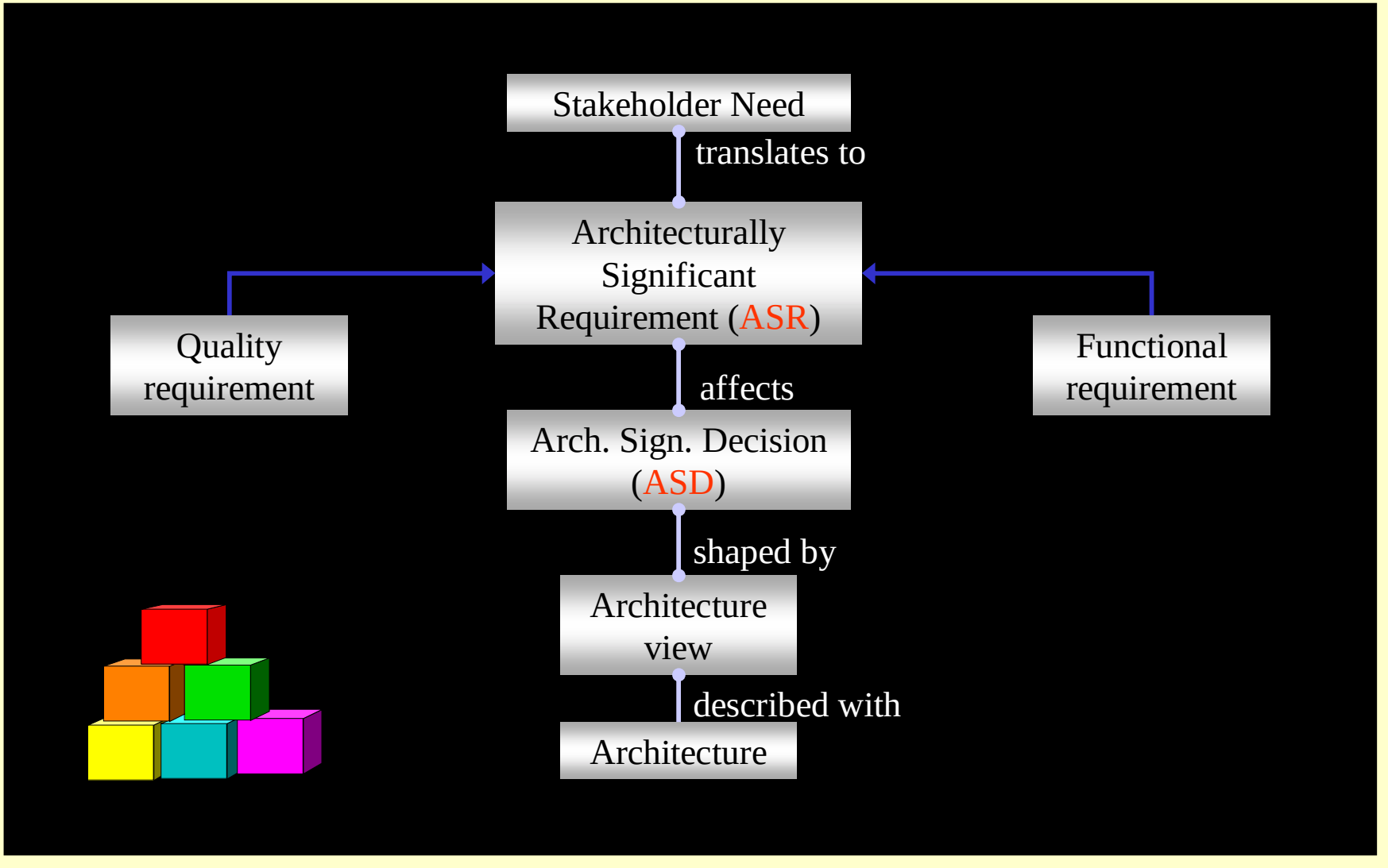
# Purpose of an architecture review



**The purpose of an architecture review is to understand the impact of an agreed set of architecturally significant decisions (ASD's) on an agreed set of architecturally significant requirements (ASR's)**

# Some review terminology





# ATAM© steps



- Presentation
  - Present the ATAM©
  - Present business drivers
  - Present architecture
- Investigation and Analysis (triage stage)
  - Identify architectural approaches
  - Generate quality attribute utility tree (QA profiles)
  - Analyze architectural approaches
- Testing (detailed examination stage)
  - Brainstorm and prioritize scenarios in a larger stakeholder group
  - Analyze architectural approaches
- Reporting
  - Present results

# ATAM© key concepts



**ATAM© identifies *risks*, *sensitivity points* and *tradeoff points***

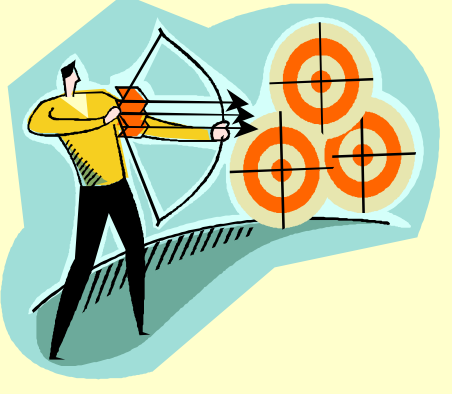
- **Risk: Architecturally important decision that has not been made or decision that has been made but whose consequences are not fully understood**
- **Sensitivity point: Parameter in the architecture to which some measurable quality attribute response is highly correlated**
- **Tradeoff point: Parameter of an architectural construct that hosts more than one sensitivity point and where the measurable quality attributes are affected differently**

# Benefits of architecture review



An architecture review helps to

- Identify risks and opportunities for improvement
- Improve communication and understanding between the stakeholders
- Improve the understanding of the architecture and its characteristics
- Help in making the right tradeoff decisions



# Ways to prevent incompleteness problems



***Good practices to decide <u>what</u> has to be done:***

- **Regular "quick scans" on completeness**
- **Explicit reviews, involving the stakeholders**
- **Clear boundaries for the architect's tasks**

***Good practices to ensure that it <u>is</u> done:***

- **A plan**
- **A delegation structure**
- **Explicit risk management**

# Why clear boundaries are important



***An architecture is complete if all relevant stakeholders are satisfied …***

***… but the architect is not the only one to satisfy the stakeholder needs.***

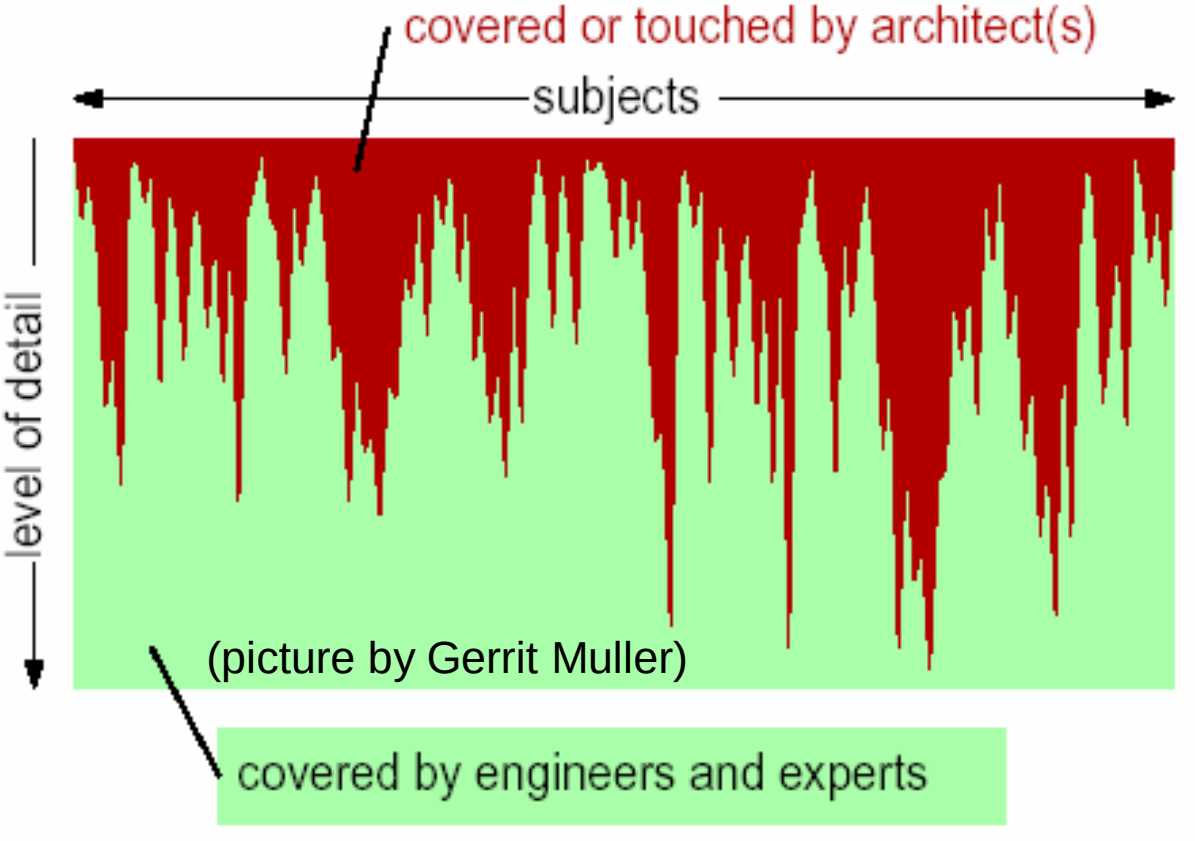
# Clear boundaries ..



**1. What subjects?**

**2a. Produce, review, check, ignore, ..?**

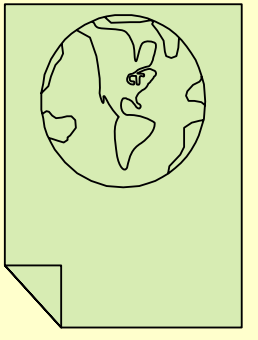
**2b. Were is the architect?**



# Defining clear boundaries



- Define boundaries in “tangible” terms, for example:
  - Along system boundaries:
    - Aggregation level / components
    - Interfaces
    - Scenarios
  - In terms of deliverables:
    - What documents?
    - What model parts?



- Make explicit what is expected from the architects
  - Using a “RACI matrix”, for example

# RACI explanation



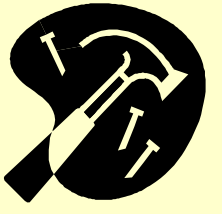
- Responsible – Architect's task is to deliver (or achieve) it; they provide the main effort
- Accountable – Architects are ultimately responsible, but do not deliver themselves (implies Consult)
- Consult – Architects either have a particular expertise they can contribute (their advice will be sought) or must be consulted for some other reason before a final decision is made (implies Inform)
- Inform – Architects are affected by the activities and decisions and therefore need to be kept informed, but do not participate in the effort

# How to ensure ACI ??



- "ACI" should be supported by the process!
  - Review
  - Test
  - Are you automatically involved when necessary ..?

- Tool support
  - Automatic code checking
    - Naming conventions
    - Automatic detection / prevention of scope violations
  - Diagnostics on resource usage, performance
  - Reverse engineering / round tripping tools



# Ways to prevent incompleteness problems



*Good practices to decide <u>what</u> has to be done:*

- Regular “quick scans” on completeness
- Explicit reviews, involving the stakeholders
- Clear boundaries for the architect’s tasks

*Good practices to ensure that it <u>is</u> done:*

- A plan
- A delegation structure
- Explicit risk management

# Benefits of (some form of) a plan



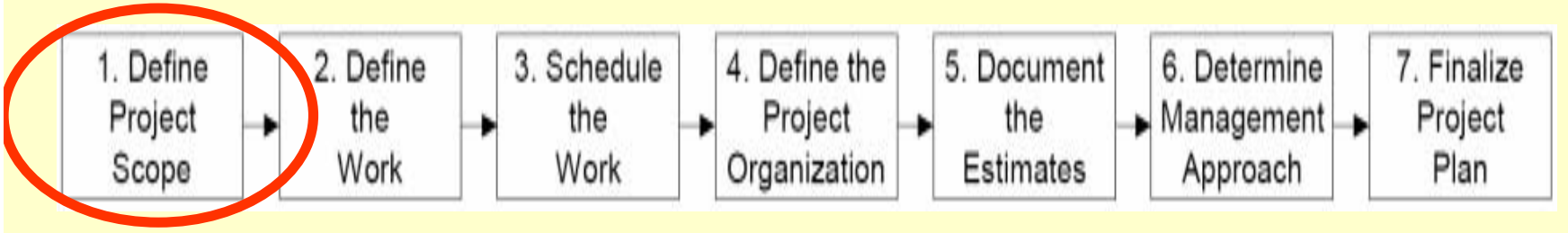
**A plan helps to prevent incompleteness:**

- **You are forced to take time to get an overview of your work (and of the risks)**
- **You have a means to get the time and resources you need**
- **You can give dependable commitments**
  - **And correct them in time**
- **Others can synchronize their activities with your plan, and vice versa**

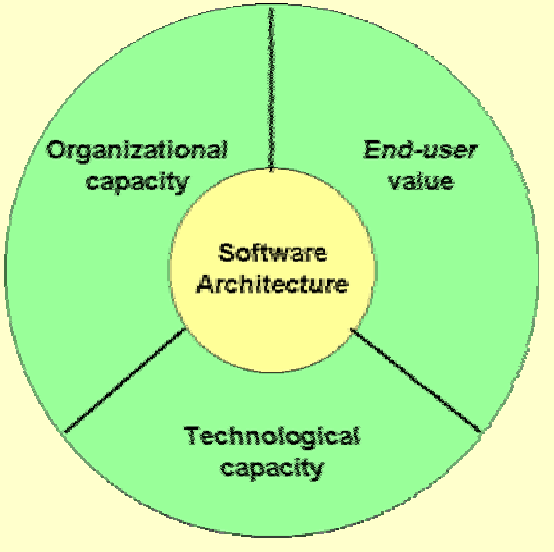
**This only works if you *maintain* your plan**

# Setting up a plan for architecture activities





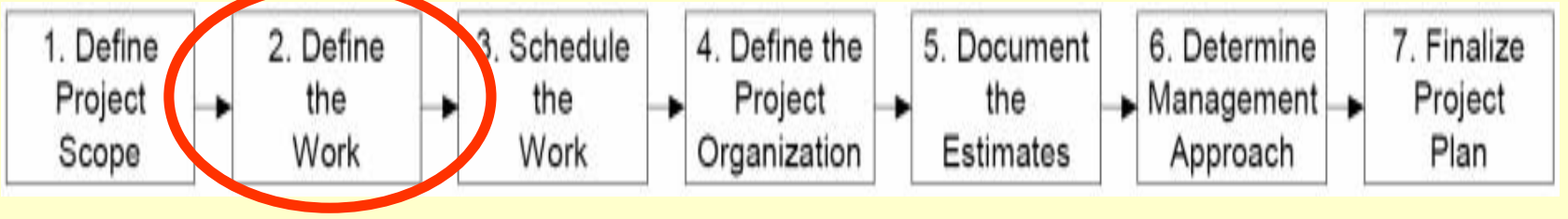

- **Who will use the software architecture for their software development?**
- **What technological environments are relevant for the software?**
- **Who will use the software "in the field"?**




- **Are all related stakeholders represented?**
- **Are the needs, expectations and constraints of these stakeholders identified and prioritized?**

# Setting up a plan for architecture activities

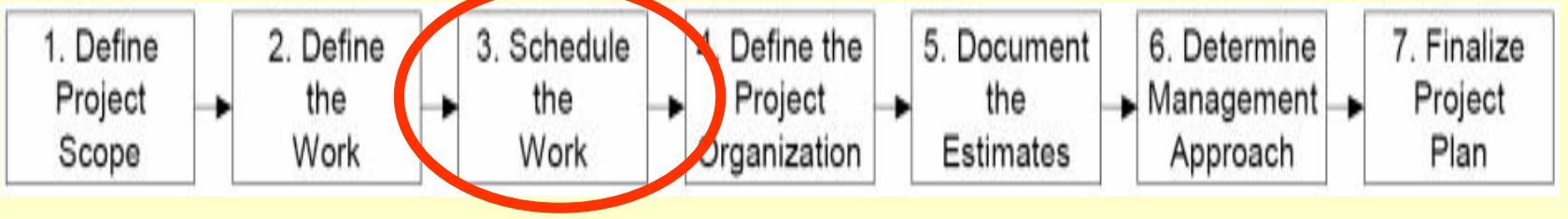





- What are the subjects to cover?
  - In what depth?
- What deliverables to develop or modify?
- What other activities?
- What are the risks and how to manage them?
  - What is the extra work involved?

# Setting up a plan for architecture activities

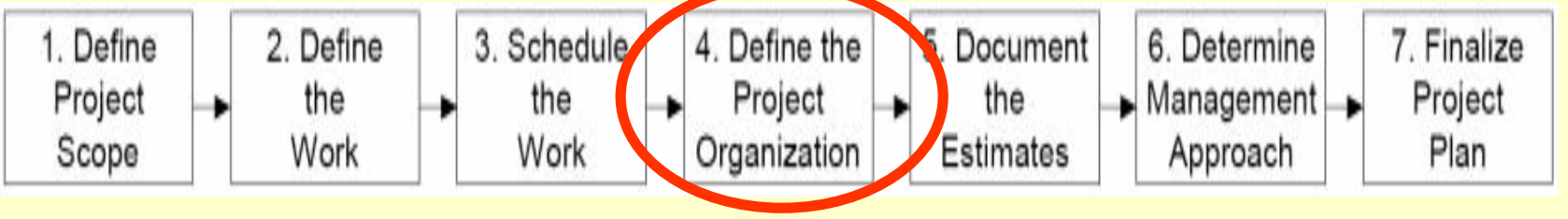





- What lifecycle to use?
  - What milestones, how many iterations, iteration length?
- What resources do we need?
- How much time will each activity take?
- What are our schedule dependencies?
- What are the priorities?
  - Based on key risks and key drivers of the system
- What is the best order of activities?
- What is the best schedule?

# Setting up a plan for architecture activities

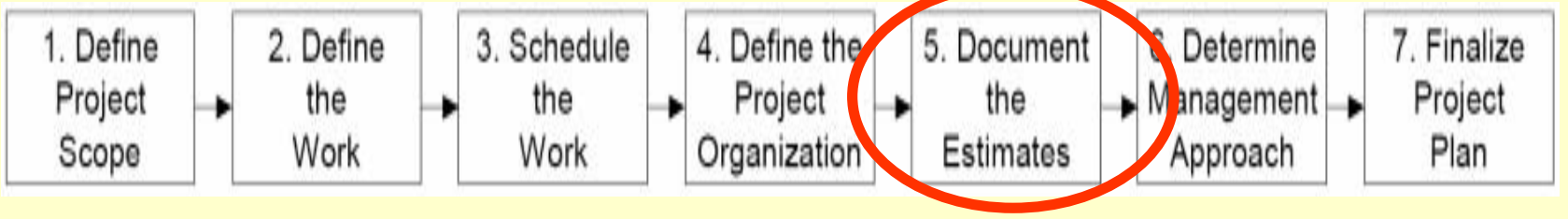





- Who is involved?
- What are their roles and responsibilities?
- Who makes what decisions and how?
- How to report and escalate?

# Setting up a plan for architecture activities

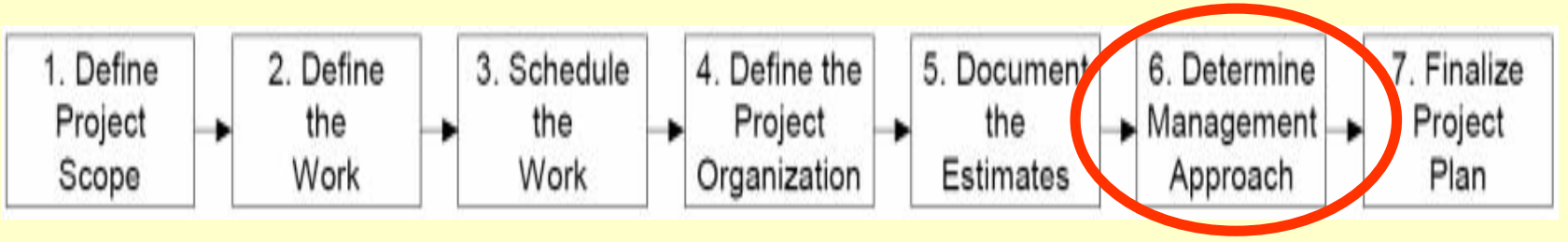





- What were the models we used for our “guestimates”?
- What were the assumptions and parameters used?
- What are “sensitivity points” for the estimations?

# Setting up a plan for architecture activities

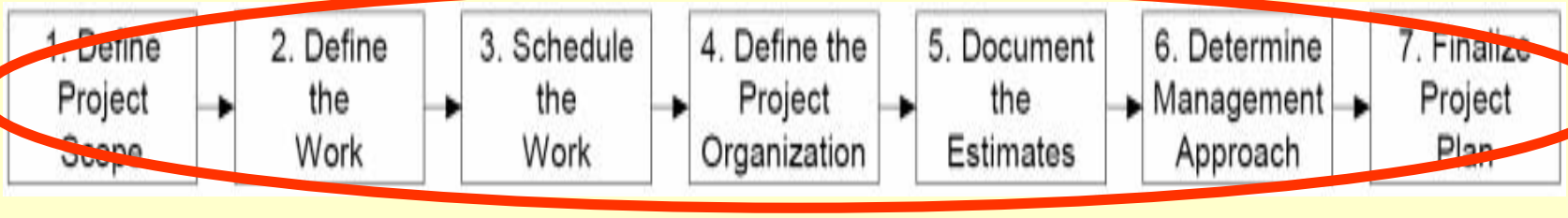





- When to re-plan or re-estimate?
  - How to measure and track the “estimation sensitivities”?
- When to plan what in detail?
- How to manage change?
- How to control quality?

# Setting up a plan for architecture activities






- **Document it**
- **Communicate it**
- **Get support and commitment for it**
- **Maintain it**

# Ways to prevent incompleteness problems



***Good practices to decide <u>what</u> has to be done:***

- **Regular "quick scans" on completeness**
- **Explicit reviews, involving the stakeholders**
- **Clear boundaries for the architect's tasks**

***Good practices to ensure that it <u>is</u> done:***

- **A plan**
- **A delegation structure**
- **Explicit risk management**

# A delegation structure



**How to perform a lot of work in a limited lead-time?**

**Work concurrently!**

- **Architecture roles on several levels**
  - **Higher levels can delegate to lower levels**
  - **In most cases along software aggregation levels**
    - **“Subsystem architects” or “cluster architects”**
  - **Sometimes different:**
    - **Scenarios / functions**
    - **Qualities**
- **Means more planning / organization / management effort!**

# Ways to prevent incompleteness problems



***Good practices to decide <u>what</u> has to be done:***

- **Regular "quick scans" on completeness**
- **Explicit reviews, involving the stakeholders**
- **Clear boundaries for the architect's tasks**

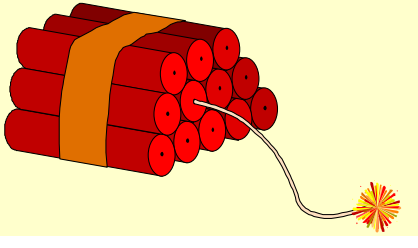
***Good practices to ensure that it <u>is</u> done:***

- **A plan**
- **A delegation structure**
- **Explicit risk management**

# Explicit risk management



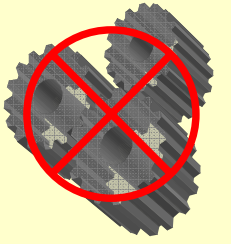
- Risks often known by architects, but possible effects are hardly communicated
  - Only cause is communicated
  - Assumption at architects is that possible effects are known
- This leads to miscalculations of project manager
  - Risk is not managed
  - No commitment to provide extra time/resources
  - This leads to an incomplete architecture
- Architects often have difficulty to translate design flaws into concrete effects



# Explicit risk management



- Create an explicit risk list
- Document the *effects* of each risk, in the frame-of-reference of the stakeholders
  - Show what will *happen*, not what is *wrong*
  - If possible, quantify the effects
    - back-of-envelope calculation is often better than nothing
- List the preventive and corrective actions
  - If possible, quantify these also!
    - back-of-envelope calculation is often better than nothing
- Maintain this list and use it in your reports
- Get the time and resources needed to tackle these risks



# Propositions



1. **Most software consists of two parts:**
   - **A part that was designed by the architects**
   - **A part that should have been designed by the architects**

   **The latter part often leads to the biggest problems..**

2. **“Completeness” has a different meaning for every system, but the practices to prevent incompleteness are very similar**