IT Architecture

Requirements, Risks, Reviews (3R)

Dr. Marcel Schlatter
IBM Distinguished Engineer
Member of the IBM Academy of Technology
marcel.schlatter@ch.ibm.com

Requirements – A Decisive Factor for Success or Failure

Top Five Project Success Factors

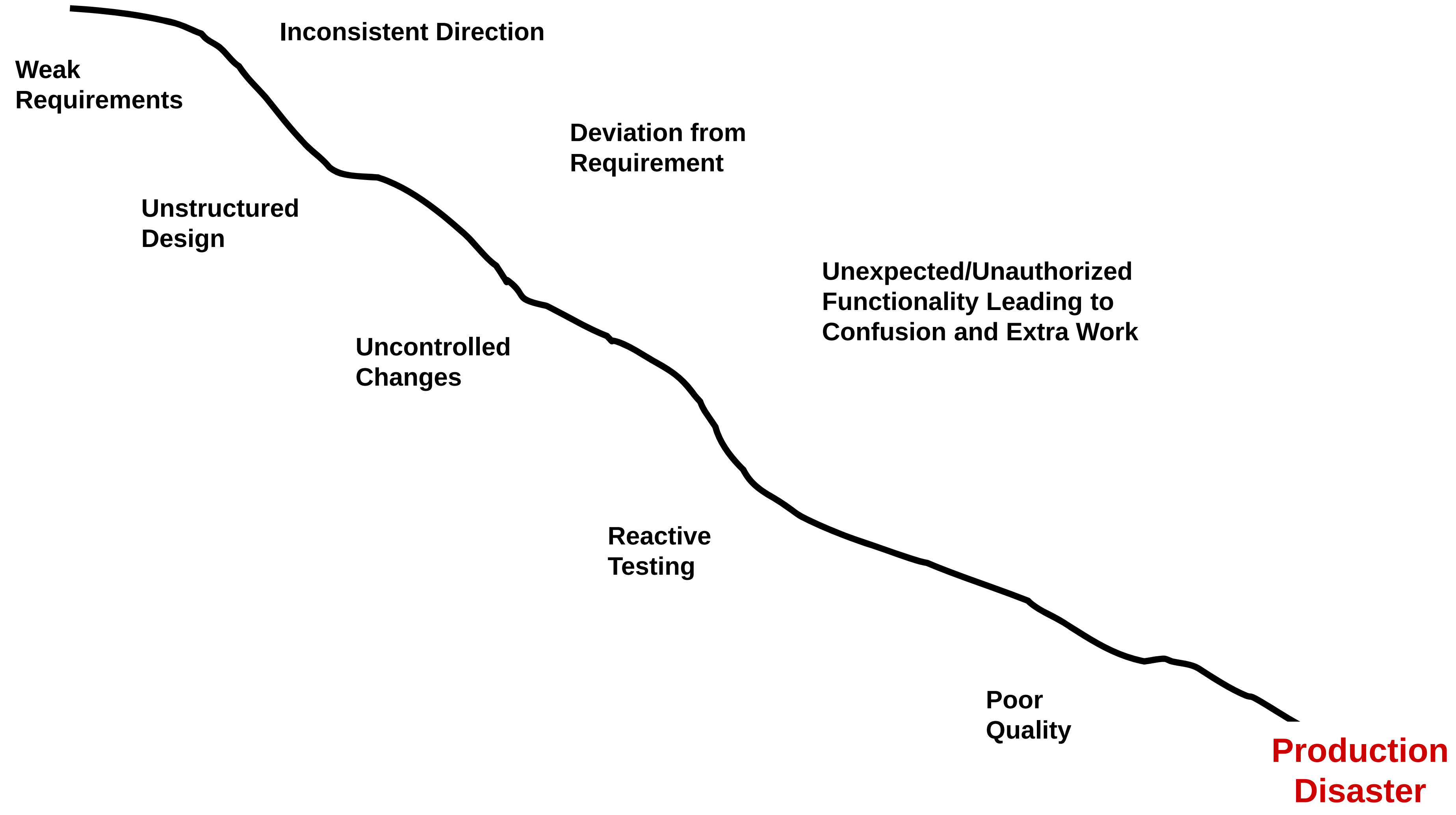
1. User Involvement
2. Executive Management Support
3. Clear Statement of Requirement
4. Proper Planning
5. Realistic Expectations

Top Ten Causes of Failed Projects

1. Incomplete Requirements
2. Lack of User Involvement
3. Lack of Resources
4. Unrealistic Expectations
5. Lack of Executive Support
6. Changing Requirement and Specifications
7. Lack of Planning
8. Didn't need it anymore
9. Lack of IT Management
10. Technical Illiteracy

Source: Standish Group

Impacts of Poor Requirements



Beware of requirements !

- Don’t assume that the original statement of the problem is necessarily the best, or even the right, one.
- Extreme requirements, expectations, and predictions should remain under challenge throughout system design, implementation, and operation.
- Explore the situation from more than one point of view. A seemingly impossible situation might suddenly become transparently simple.
- The most important single element of success is to listen closely to what the client perceives as his requirements and to have the will and ability to be responsive.
- Success is defined by the client, not by the architect.

A lot of times, people don't know what they want until you show it to them.

Steve Jobs

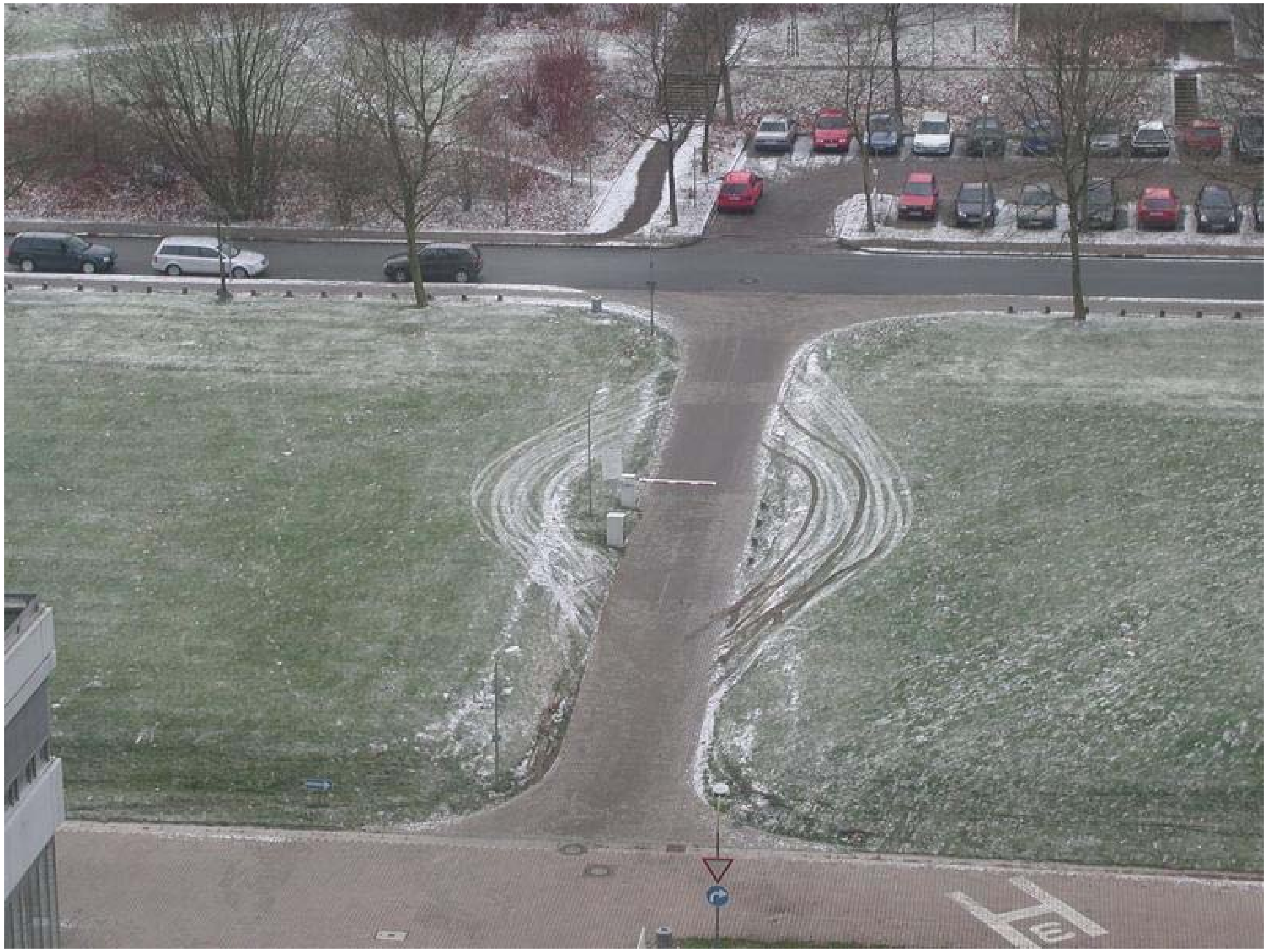
Beware of requirements !

- Don’t assume that the original statement of the problem is necessarily the best, or even the right, one.
- Extreme requirements, expectations, and predictions should remain under challenge throughout system design, implementation, and operation.
- Explore the situation from more than one point of view. A seemingly impossible situation might suddenly become transparently simple.
- The most important single element of success is to listen closely to what the client perceives as his requirements and to have the will and ability to be responsive.
- Success is defined by the client, not by the architect.

A lot of times, people don't know what they want until you show it to them.

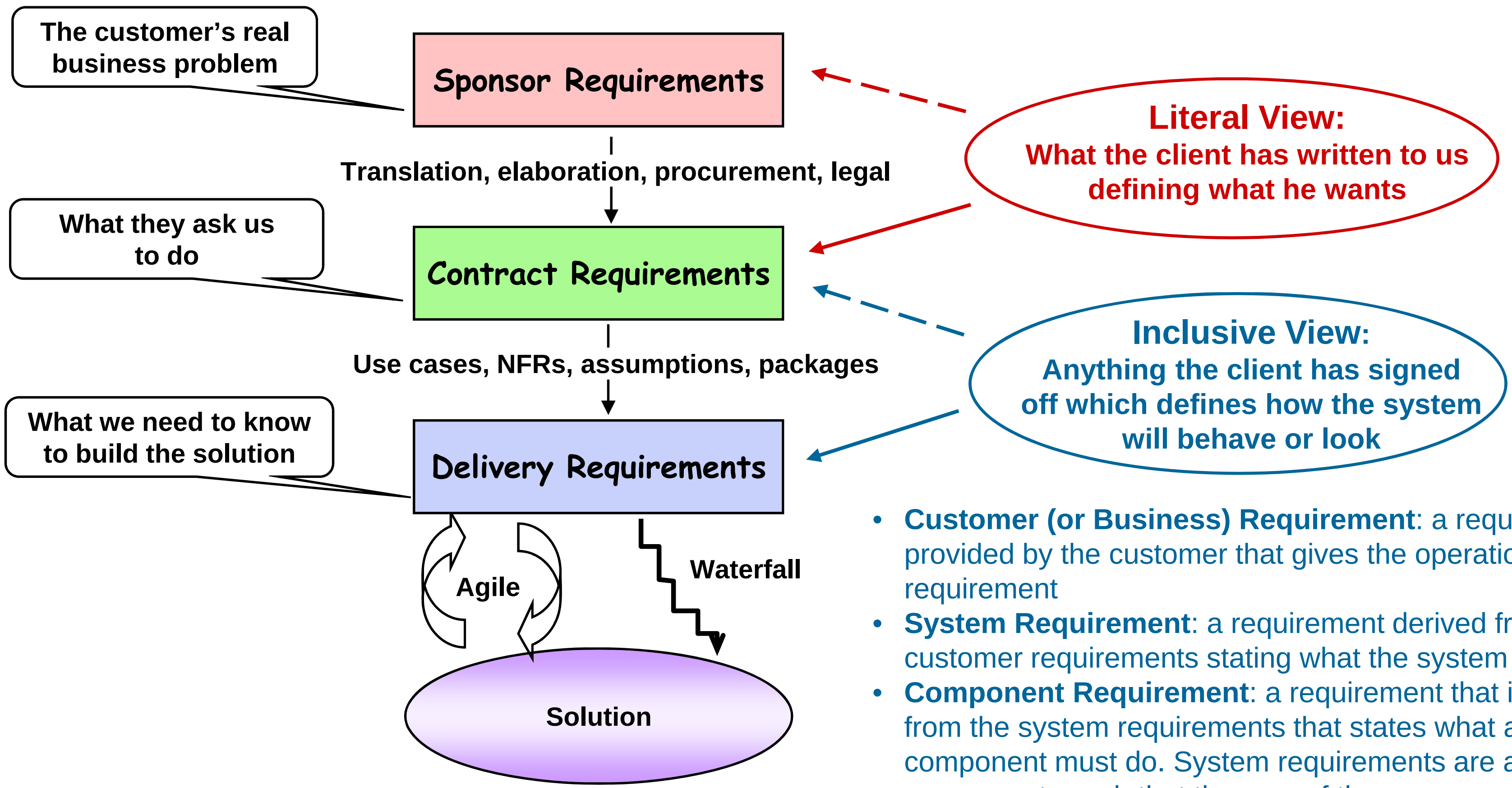
Steve Jobs

Don’t assume that the original statement of the problem is necessarily the best, or even the right one.



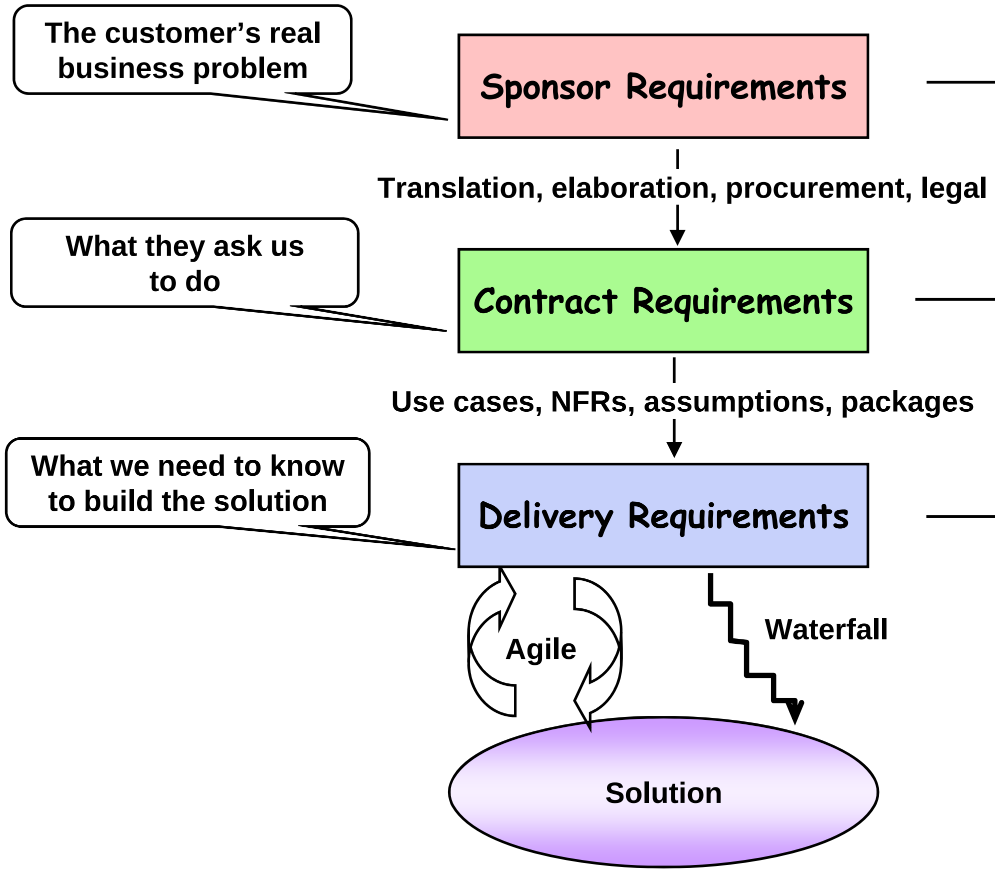
Requirements Realities
Types of Requirement…

What is a Requirement? Two different views…


The customer's real business problem
Sponsor Requirements
Translation, elaboration, procurement, legal
What they ask us to do
Contract Requirements
Use cases, NFRs, assumptions, packages
What we need to know to build the solution
Delivery Requirements
Agile
Waterfall
Solution
Literal View:
What the client has written to us defining what he wants
Inclusive View:
Anything the client has signed off which defines how the system will behave or look

- **Customer (or Business) Requirement**: a requirement provided by the customer that gives the operational requirement
- **System Requirement**: a requirement derived from the customer requirements stating what the system must do
- **Component Requirement**: a requirement that is derived from the system requirements that states what a component must do. System requirements are allocated to components such that the sum of the component requirements fully satisfies the system requirement

Requirements Realities
Types of Requirement… Each Type is Important…


The customer's real business problem
Sponsor Requirements
Translation, elaboration, procurement, legal
What they ask us to do
Contract Requirements
Use cases, NFRs, assumptions, packages
What we need to know to build the solution
Delivery Requirements
Agile
Waterfall
Solution

Must understand and manage, else…

- Customer dissatisfaction
- Project cancelled or no new work
- Don't get paid
- Can't build and test in orderly way

Requirements are the foundation …

- Requirements form the basis for design, development, test, and operations.
- Each requirement has a cost, technical, or schedule impact.
- It is essential that a complete, but minimum set of requirements be established and maintained as the basis.
- **You cannot ignore requirements problems…**

Chief Architects Overarching Goal… Insuring Requirements Quality…

A Framework for Assessing Requirements Quality & Identifying Issues…

1. **Start with the Sponsor & Contract Requirements…**
2. **Validate your understanding of Sponsor & Contract Requirements…Conditions of Satisfaction…Priorities…**
3. **Then follow thru with an analysis of the System & Component Requirements…**

For each level of requirements. determine if the requirements are:

- **Unambiguous & Understandable** – every requirement has only one interpretation…the interpretation of each requirement is clear
- **Verifiable** – a finite, cost-effective process has been defined to check that the requirement has been attained
- **Concise** – no unnecessary information is included in the requirement
- **Unique** – requirement(s) is (are) not overlapping or redundant with other requirements
- **Complete** – (a) everything the system is required to do throughout the system's life cycle is included, (b) responses to all possible (realizable) inputs throughout the system's life cycle are defined, (c) the document is defined clearly and self-contained, and (d) there are no to be defined (TBD) or to be reviewed (TBR) statements; completeness is a desired property but cannot be proven at the time of requirements development, or perhaps ever
- **Comparable** – the relative priority of the requirements is included
- **Attainable** – solutions exist within performance, cost and schedule constraints

Leadership Goal…
Eliminate Requirements Ambiguity and Ensure Clarity and Testability…

- As the Chief Architect, you must turn **ambiguity** into requirements that are clear and testable.
- If you can not fix the requirements themselves…ensure you have definitive statements placed into the **acceptance criteria**.
- If you can not **test** a requirement, it is meaningless…

- More broadly, watch out when the customer is introducing **ambiguity**.
- Phrases like “Shall not be limited too”, “including but not limited too”, “various”,… are very dangerous!

Leadership Goal…
Eliminate Requirements Ambiguity and Ensure Clarity and Testability…

Dealing with **Vague** and **Ambiguous** Requirements

- Do not use subjective and ambiguous words
  - Acceptable, Adequate, Sufficient
  - Better, Superior
  - Efficient, Fast, Flexible, Robust, Seamless, Graceful
  - Maximize, Minimize, Optimize
  - Normally, Reasonably
  - Shouldn't
  - State-of-the-Art, User-friendly, Easy, Simple
- Resolve TBD's → If absolutely necessary, include a description of what must be done to eliminate the TBD, who is responsible, and by when
- Have a diverse team review the requirements…
- Develop prototypes, system models (DFDs, ERDs,…)
- **Develop test plans and test cases as early as possible**

Chief Architects Overarching Goal…
Insuring Requirements Quality…

A Framework for Assessing Requirements Quality & Identifying Issues…

1. **Start with the Sponsor & Contract Requirements…**
2. **Validate your understanding of Sponsor & Contract Requirements…Conditions of Satisfaction…Priorities…**
3. **Then follow thru with an analysis of the System & Component Requirements…**

For each level of requirements. determine if the requirements are:

- **Unambiguous & Understandable** – every requirement has only one interpretation…the interpretation of each requirement is clear
- **Verifiable** – a finite, cost-effective process has been defined to check that the requirement has been attained
- **Concise** – no unnecessary information is included in the requirement
- **Unique** – requirement(s) is (are) not overlapping or redundant with other requirements
- **Complete** – (a) everything the system is required to do throughout the system's life cycle is included, (b) responses to all possible (realizable) inputs throughout the system's life cycle are defined, (c) there are no to be defined (TBD) or to be reviewed (TBR) statements; completeness is a desired property but cannot be proven at the time of requirements development, or perhaps ever
- **Comparable** – the relative **priority** of the requirements is included
- **Attainable** – solutions exist within performance, cost and schedule constraints

Ensure Requirements Completeness…
Manage Customer Expectations…
Ensure a Clear and Shared View of Priorities…

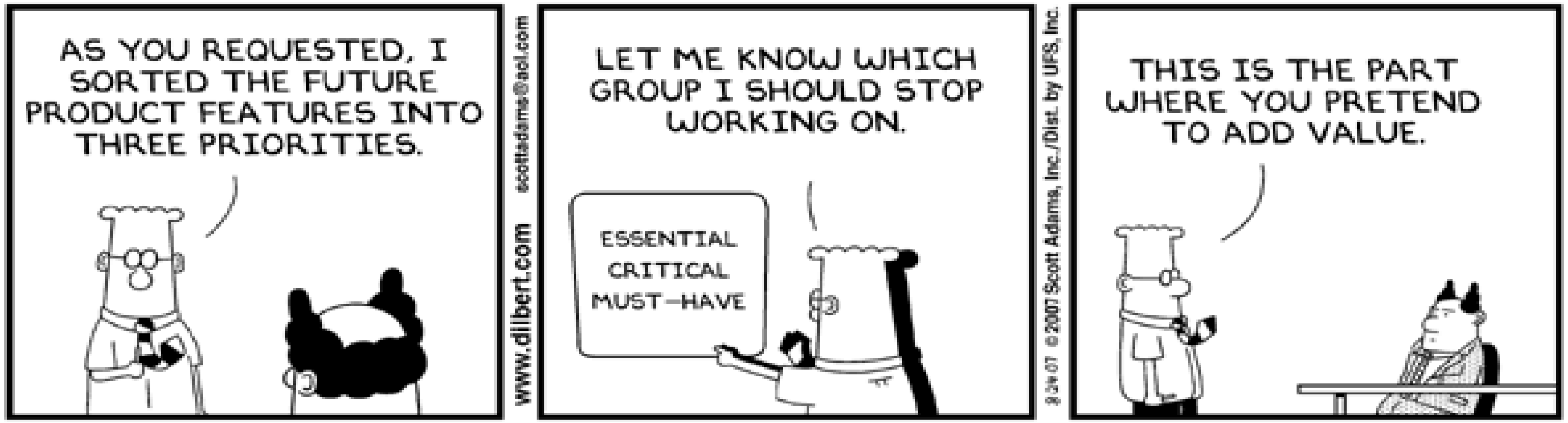
- Chief Architect must ensure that Requirements are complete…
  - Incomplete Requirements lead to customer dissatisfaction, cost and schedule overruns…
- The Customer is often unsure…
  - Decide what you will provide them
  - Document your assumptions / decision
  - Publish it to the customer (paper / demo / etc.)
- Insure complete alignment between requirements and customer expectations

- Requirements must be prioritized…
  - Sponsor and other stakeholder needs should drive the prioritization…
- Prioritizing requirements enables efficient project team responses to:
  - New requirements added during development
  - Budget cuts
  - Schedule overruns
- Priority is a function of the value provided to the customer, the relative cost of implementation, and the relative technical risk associated with implementation.

Ensure a Clear and Shared View of Priorities…
IEEE Standard 830-1998

- **High** - Solution not acceptable unless these requirements are provided.
- **Medium** - Requirements that enhance the product, but would not make the product unacceptable if they are absent.
- **Low** – Other requirements.

Prioritizing Requirements: Antipattern


AS YOU REQUESTED, I SORTED THE FUTURE PRODUCT FEATURES INTO THREE PRIORITIES.
www.dilbert.com scottadams@aol.com
LET ME KNOW WHICH GROUP I SHOULD STOP WORKING ON.
ESSENTIAL
CRITICAL
MUST-HAVE
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.
THIS IS THE PART WHERE YOU PRETEND TO ADD VALUE.

Chief Architects Overarching Goal…
Insuring Requirements Quality…

A Framework for Assessing Requirements Quality & Identifying Issues…

1. **Start with the Sponsor & Contract Requirements…**
2. **Validate your understanding of Sponsor & Contract Requirements…Conditions of Satisfaction…Priorities…**
3. **Then follow thru with an analysis of the System & Component Requirements…**

For each level of requirements. determine if the requirements are:

- **Unambiguous & Understandable** – every requirement has only one interpretation…the interpretation of each requirement is clear
- **Verifiable** – a finite, cost-effective process has been defined to check that the requirement has been attained
- **Concise** – no unnecessary information is included in the requirement
- **Unique** – requirement(s) is (are) not overlapping or redundant with other requirements
- **Complete** – (a) everything the system is required to do throughout the system's life cycle is included, (b) responses to all possible (realizable) inputs throughout the system's life cycle are defined, (c) the document is defined clearly and self-contained, and (d) there are no to be defined (TBD) or to be reviewed (TBR) statements; completeness is a desired property but cannot be proven at the time of requirements development, or perhaps ever
- **Comparable** – the relative priority of the requirements is included
- **Attainable** – solutions exist within performance, cost and schedule constraints

Leadership Goal…

Ensure Requirements Feasibility…

What is a "requirement"

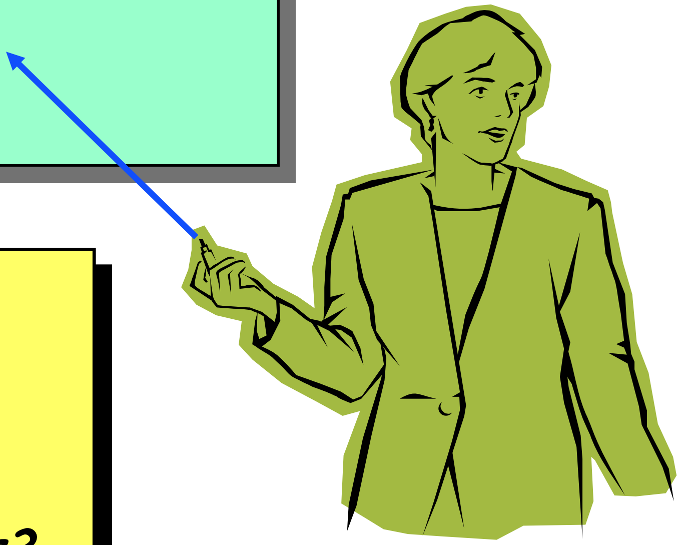
in <u>your</u> world?

<u>Is it still a requirement if…</u>

…it costs too much?

…it is not technically feasible?

What do you do in these situations?



Leadership Goal…
Ensure Requirements Feasibility…and Reasonable…

- Chief Architect must ensure that it is feasible to implement each requirement.
- Chief Architect must believe there is a reasonable way to achieve each requirement…sometimes this is a judgement call.
- To avoid infeasible requirements, delivery team involvement is desirable.
- Inputs from the delivery teams should be solicited and issues that are raised should be addressed.

Chief Architects Overarching Goal…
Insuring Requirements Quality…

A Framework for Assessing Requirements Quality & Identifying Issues…

Within and between levels of requirements, determine if the requirements are:

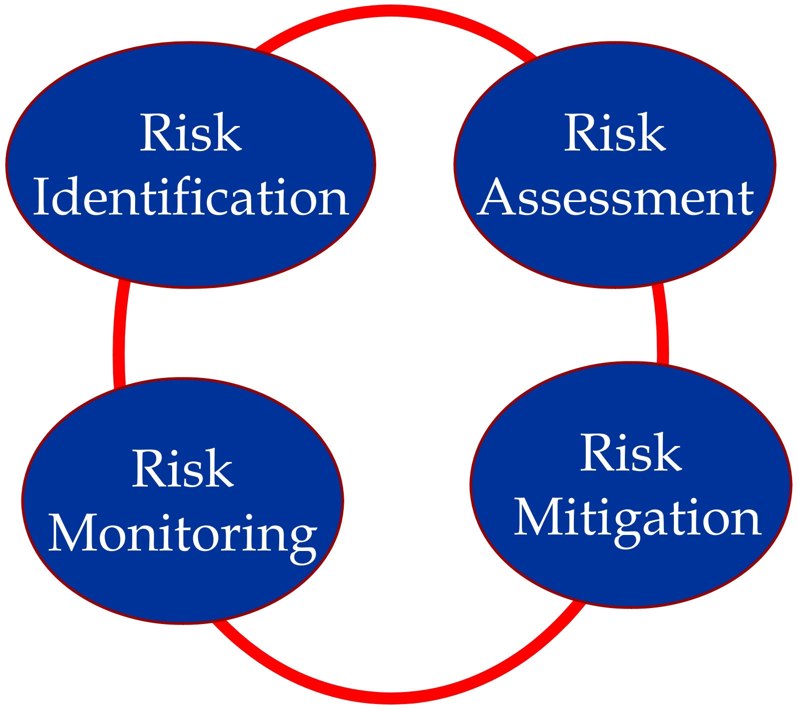
- **Traced**: each requirement is traced to some document or statement of the stakeholders
- **Traceable**: each derived requirement must be traceable to an originating requirement
- **Consistent**:
  - **Internal:** no two subsets of requirements conflict
  - **External**: no subset of requirements conflicts with external documents from which the requirements are traced

Risks

How Can You Manage Risks?

Risk Management =
Maintaining and allocating resources to ensure that future events do not jeopardize project success.

- Risk identification
  - What should you care about?
- Risk assessment
  - Why should you care about them?
- Risk mitigation
  - What will you do about them?
- Risk Monitoring
  - How will you status the project?



Handle Risks

What Techniques Do You Commonly Use?

- Develop alternatives and plans to handle top priority risks and all risks above their thresholds
- Method for handling risks include:
  - Assume the risk and fix on failure
  - Assume the risk but plan contingencies
  - Avoid the risk by elimination
  - Reduce likelihood and / or consequences
  - Investigate further before deciding
  - Transfer the risk

- Include Proofs of Concept and prototypes
- Partition Solution Development & Integration
- Prioritize deliverable content and phase client deliverables
- Implement an early and aggressive integration and test schedule
- Identify backup suppliers (for skills or technologies)
- Include an interval to accommodate potential slips
- Use rigorous change management processes
- Establishing Technical Performance Metrics

Dealing with REALITY

- There’s not enough staff time (human hours) or schedule time to address all potential risks.
- Which risks are unacceptable?
- Can we avoid or mitigate these?

Can we live with what we can’t fix?

Will the mitigation strategy work?

Reviews

As the author of the solution, the architect is undeniably accountable for the effort's success or failure.

Reviews cannot take that responsibility away from the architect, but reviews can help to identify risks that the architect may have overlooked, things that may be missing, or bad architectural

- **Topics** that the Lead Architect typically covers in a presentation at the beginning of a review
- **Qualities** that are typically reviewed
- **Areas of risk** that are typically investigated in a review

Topics that the lead architect typically covers in a presentation at the beginning of the review (1 of 3)

- **Driving architectural requirements**
  - Measurable quantities associated with these requirements
  - Any existing standards / models associated with these requirements
  - Any existing standards / models / approaches for meeting these requirements

- **High-level architectural views**
  - Other systems with which the system must interact
  - Key events which the architecture must enable
  - System overview
  - Inter-dependencies of the sub-systems
  - Technical constraints such as an operating system, hardware, or middleware prescribed for use
  - Architectural approaches used to meet quality attribute requirements
  - Key workproducts, covering requirements, performance, availability, capacity, security, manageability, development, testing, and implementation

- **Architectural issues / risks** with respect to meeting the driving architectural requirements

- **Functional view**: Functions, Business Rules, Data Flows

Topics that the lead architect typically covers in a presentation at the beginning of the review (2 of 3)

- **Service Model (SOA)**
  - Service identification: Online requests, non-interactive requests, Batch “Mass Requests”
  - Relationship between business processes and services
  - Service specification: Service Versioning, Message Versioning and Schema Validation
  - Service catalog
  - Service hierarchy
  - End-to-end view: showing how information flows from system border (service interface) to the backend and vice versa – bridging all system, layer, protocol, technology, etc. borders

- **Code:** the system’s decomposition of functionality
  - Subsystems
  - Layers
  - Components
  - Modules (Objects, procedures, and functions that populate the modules)
  - Relationships among modules (procedure call, method invocation, callback, containment)
  - Concurrency (Processes, thresds, synchronizatoin, dataflow, events that connect processes and threads)

- **Physical:** CPU’s, Storage, Networks and communication devices that connect them

- **Architectural approaches**, styles, patterns, or mechanisms employed
  - What quality attributes are addressed by these approaches, styles and patterns
  - Description of how the approaches address those attributes

Topics that the lead architect typically covers in a presentation at the beginning of the review (3 of 3)

- **Trace** of 1 – 3 of the most important use case scenarios
  - including the run-time resources consumed for each scenario
- **Trace** of 1 – 3 of the most important growth or other change scenarios, describing the change impact
  - estimated size / difficulty of the change in terms of the changed components, connectors, services, and interfaces
- **Process description**
  - Problem management
  - Change management
  - Operational change control
  - Defect management
  - Release management
  - Test and integration
  - Deployment
  - Acceptance
  - Document management
- **Assessment** of the lead architect: challenges, issues, and areas of concern
  - Technical risk register and technical roadmap
  - Is the system architecture documentation fit for the purpose in terms of completeness, level of detail, currency and structure?

Qualities that are typically reviewed (1 of 2)

- **Functionality**: Ability of the system to do the work for which it is intended
  - Requires that the system's components work in a coordinated manner to complete the job
- **Performance**: Responsiveness of the system
  - Time required to respond to stimuli (events)
  - Number of events processed in some interval of time
- **Reliability**: Ability of the system to keep operating over time
  - Measured by mean time to failure
- **Availability**: Proportion of time the system is up and running
  - Measured by the length of time between failures, and how quickly the system is able to resume operation in the event of failure
- **Security**: Ability to resist unauthorized attempts at usage and denial of service while still providing its services to legitimate users
  - Categorized in terms of the types of threat that might be made to the system

Qualities that are typically reviewed (2 of 2)

- **Modifiability**: Ability to make changes to a system quickly and cost effectively
  - Ability to make isolated changes
  - Measured by using specific changes as benchmarks and recording how expensive those changes are to make
- **Conceptual integrity**: underlying theme or vision that unifies the design of the system at all levels
  - The architecture should do similar things in similar ways
  - The architecture should exhibit consistency
  - The architecture should have a small number of data and control mechanisms
  - The architecture should use a small number of patterns to get the job done
- **Operation of technical management** across the system life-cycle
  - Requirements traceability (functional and nonfunctional)
  - Architectural governance
  - Implementation governance

Areas of risk that are typically investigated in a review (1 of 3)

If the answer to any of the questions is „no“ then this indicates an area of risk which needs to be mitigated

- Requirements
  - Have likely changes been identified and recorded?
- Architecture
  - Is it possible to map, or decompose, requirements to individual elements in the architecture?
    - Functional requirements
    - Quality attributes: Modifiability, Performance, Security, Availability, Reliability
  - Can the architecture accommodate the identified changes?
- Detailed design and implementation
  - Are notations and programming systems used which facilitate verification that the software meets its requirements?

Areas of risk that are typically investigated in a review (2 of 3)
If the answer to any of the questions is „no“ then this indicates an area of risk which needs to be mitigated

- Verification
  - Are all specifications, items of software, test scripts, etc. **reviewed** by people other than their authors?
  - Have the designs and items of software been analysed by tools that can detect errors and inconsistencies?
  - Are individual software components **tested** by people other than their authors?
  - Is there a **systematic** way of ensuring that all the software in individual components is thoroughly **tested**?
  - Is there a way of **systematically testing** the software as it is **progressively integrated** to produce the complete system?
  - Is there a means of **regression testing** following changes (to ensure that changes have not caused unintended changes in system behavior)?
- Traceability
  - Is all software traceable to a top-level requirement?
  - Are all requirements traceable to the software which implements them?

Areas of risk that are typically investigated in a review (3 of 3)

If the answer to any of the questions is „no“ then this indicates an area of risk which needs to be mitigated

- Configuration management
  - Is it possible to identify every software item in a particular build, or release, of the software system?
  - Is it possible to re-build any previous release of a software system?
  - Is it possible to know to which item of software a user problem report relates?
- Change management
  - Is it possible to determine the impact on the software and all associated information (requirements, architecture, tests) of any proposed change before implementing it?
  - Are changes grouped to minimise their impact and to ease their verification?
- Things missing (NFR)
  - Plausibility checks?
  - Dealing with unexpected input, unexpected return codes from the infrastructure, etc.?
  - Application monitoring instrumentation?
  - Queue monitoring?
  - etc.