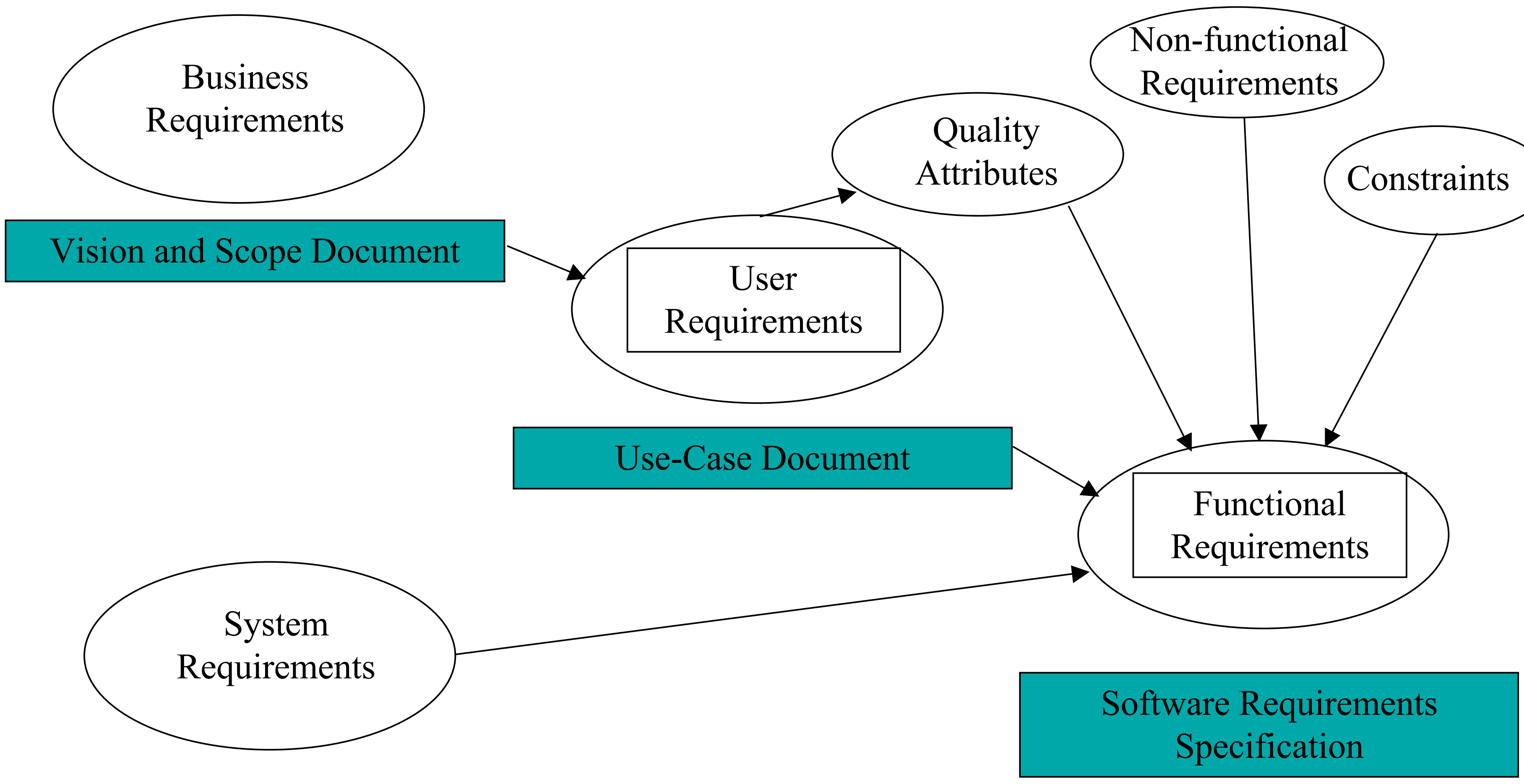
Ten Requirements Traps (Wiegers)

1. Confusion over “requirements”
2. Inadequate customer involvement
3. Vague and ambiguous requirements
4. Unprioritized requirements
5. Building functionality no one uses
6. Analysis paralysis
7. Scope creep
8. Inadequate change process
9. Insufficient change impact analysis
10. Inadequate version control

Confusion over “requirements”

- customer provided reqts often are really solution ideas
- reqts discussions focus exclusively on functionality
  - A real project needs other stuff:
    - business reqts - high level business objectives
    - user reqts - interactions between users and system
    - functional reqts - specific behavior derived from use cases

How do you develop software requirements?



Inadequate Customer Involvement

- Often users aren’t all that involved
- Need to
  - identify user classes
    - gain direct info from each class
  - make use of a “product champion”

Vague and Ambiguous Requirements

- Ambiguity - several possible meanings
  - worst: multiple interpretations go undetected
  - can’t build simple black box test cases
  - developers (designers) have to ask questions
- Solutions
  - ***avoid subjective and ambiguous language!***
  - ***write test cases early***
  - ***formal document inspection with various perspectives***
  - ***prototype and build alternative models***

Uprioritized Requirements

- Or all (most) of “very high” priority
  - problems during the “descoping phase” later :-)
- Prioritize on user’s needs
  - or frequency of use
  - favored user class
  - core business process
  - legal/regulatory constraints
- Use at least 3 priority levels

Building Functionality No One Uses

- beware of developer “chrome” GUI if ignoring actual useful system behavior
  - functionality should be clearly related to known user tasks or business goals
- solution:
  - traceability of requirements to origins
    - use case, rule, person, standard
  - have customer rate value of each feature (and the penalty if it is not implemented)
  - risk/benefit for features is good during crunch time

Analysis Paralysis

- Documents and process need not be perfect
  - the “best” can be the enemy of the “good”
- Acceptable risk is not zero risk
- Process is there to support product
  - and product can’t be produced without process

Inadequate Change Process

- For later consideration - though it’s happened already :-)
- You can get very bogged down without a defined change process

Insufficient Change Impact Analysis

- Costs and benefits of changes analyzed

Inadequate Version Control

- Critical now
  - versioning system to distinguish drafts from baselined documents

Keys to Excellent Reqts

1. Educate developers, mngrs, customers about reqts and appl.domain
2. Establish collaborative cust-devel partnership
3. Categorize customer input into appropriate reqt category
4. Take iterative and incremental approach to reqts devel
5. Use std templates to customize for V&S, Use Case and SRS docs
6. Hold formal and informal reviews of reqts docs
7. Write test cases against reqts
8. Prioritize reqts in an analytical fashion
9. Basic discipline and “good enough” attitude to handling reqts and changes