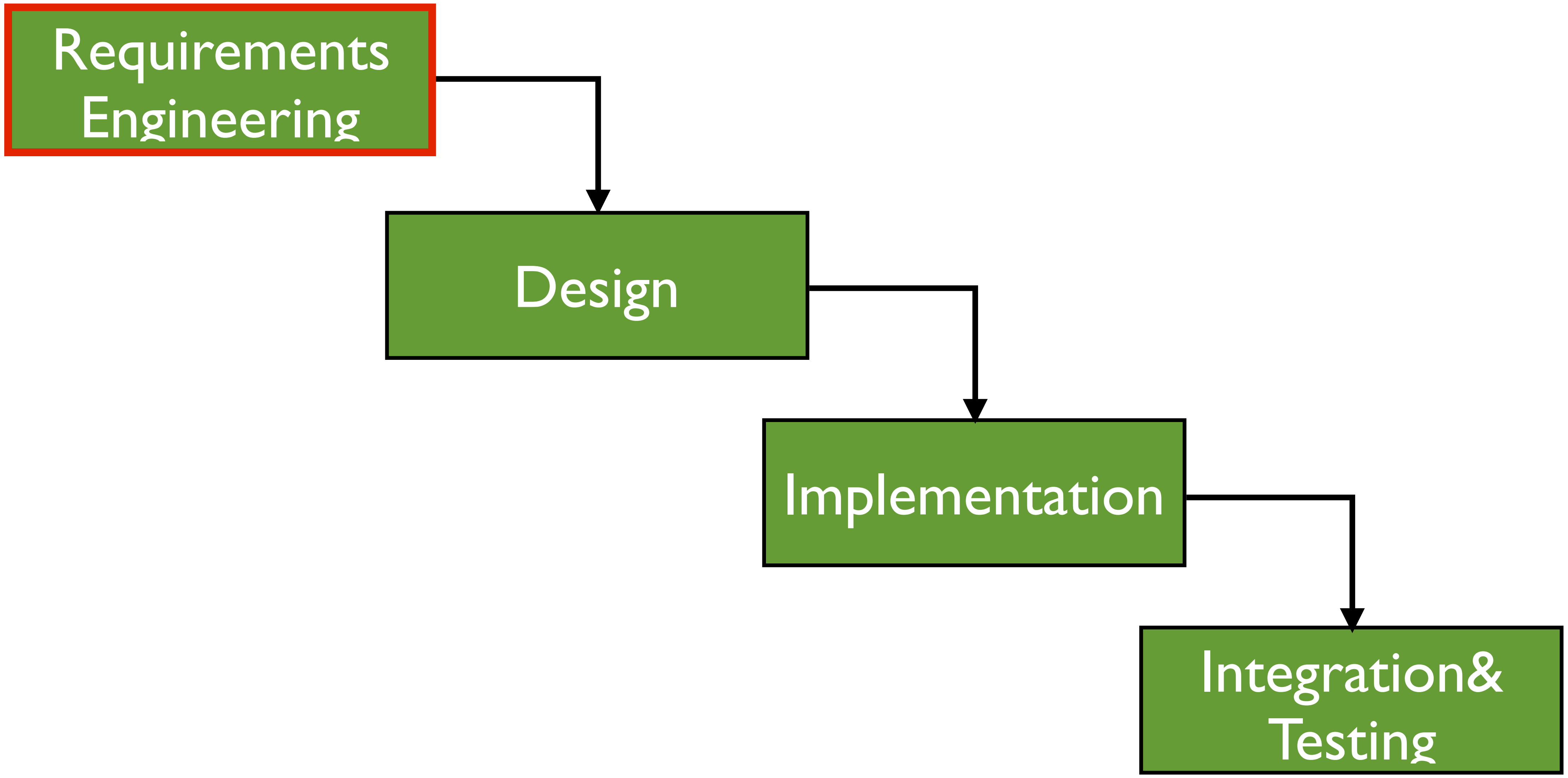
CE/CS/SE 3354 Software Engineering

Requirements Engineering

Basic Steps in Software Process Models



Requirements Engineering

- First stage of software life cycle
- Defines functionalities and constraints
- Produces a document, software requirements specification (SRS) to bridge customers and developers
    - Customers provide high level ideas
    - Software developers need a more detailed specification

Software Requirements

Requirements are means of communication with customer and other stakeholders

[by Helene Wong, PhD thesis, 1994]

Stakeholders

- People who support, benefit from, or affected by a software project
- Example: a medical system for sharing information among medical care providers
  - Who are the stakeholders?

Stakeholders

- Stakeholders may include
  - Customers
  - Users
  - Managers
  - System administrators
  - Consultants
  - Software engineers
  - ...

Types of Requirements: Functionalities

- Functional requirements
  - What is the system supposed to do
  - Mapping from input to output
- Non-functional requirements
  - Performance
  - Resource Consumption
  - Usability
  - Reliability
  - Robustness
  - Portability
  - …

Measures of Non-functional Requirements

Property	Example Measure
Performance	Processed transactions per second User/event response time
Resource Consumption	Mbytes
Usability	Training time Number of help frames
Reliability	Mean time frame between failures Rate of failure occurrence
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Types of Requirements: Constraints

- Process constraints
  - a restriction on the techniques or resources that can be used to build the system
- Design constraints
  - a design decision such as choice of platform or interface components

Types of Requirements: Example

Functional requirement: The email must be sent when requested

Non-Functional requirement: The latency should not be more than one hour

Process constraint: Two developers will be involved in this project

Design constraint: The project will be implemented for the Android platform

Requirements Are Important

The hardest single part of building a software system is deciding precisely what to build.

No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all interfaces to people, to machines, and to other software systems.

No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

[by Frederick Brooks, “No silver bullet: essence and accidents of software engineering”, 1986]

Requirements Are Important (Cont'd)

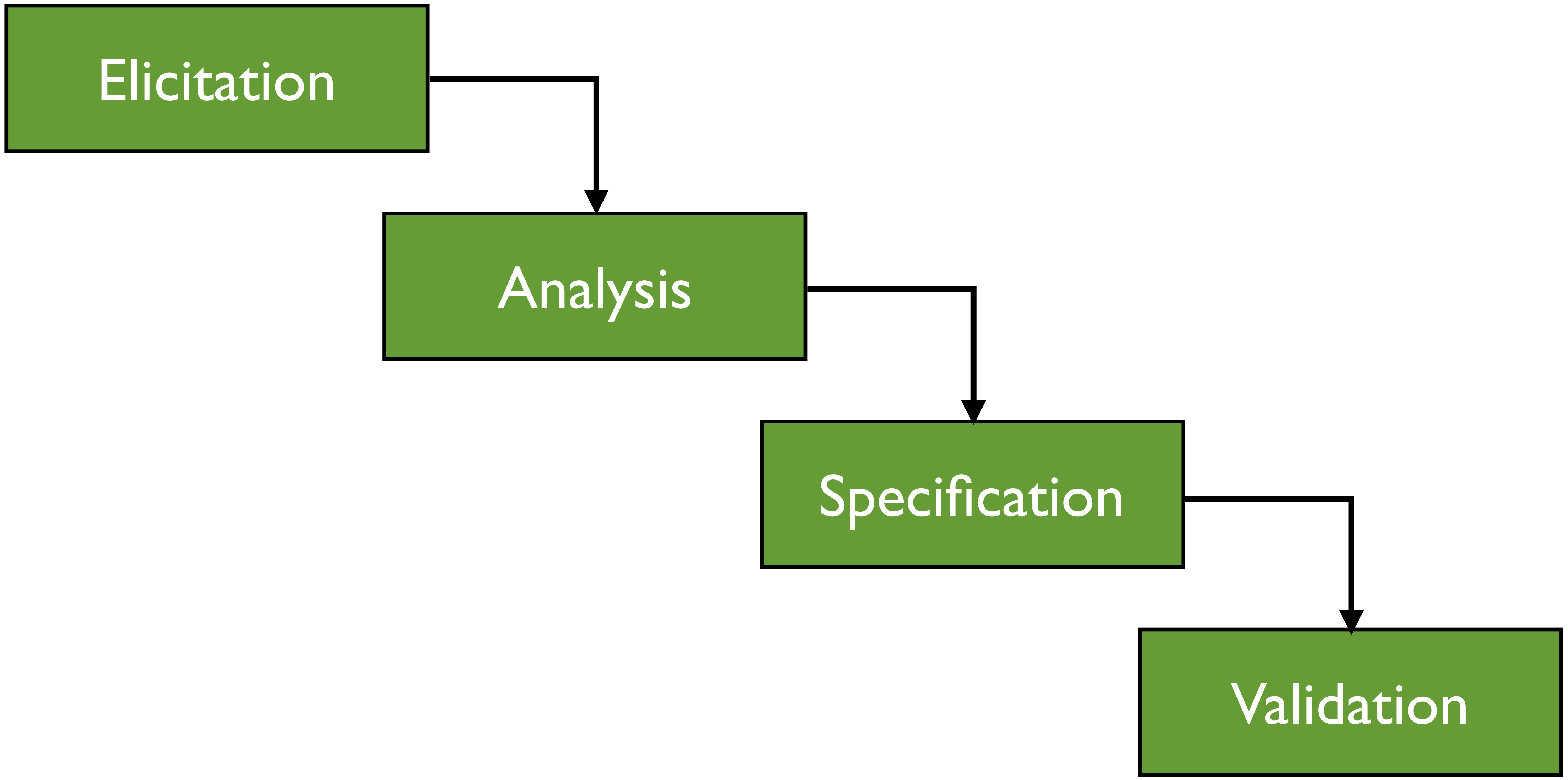
- 80% of software errors are requirements errors
  - Here software errors are defined as errors detected after unit testing – i.e., in integration testing and after the software is released
  - Most errors can be traced to unknown, wrong, or misunderstood requirements

Requirements Are Important (Cont'd)

- As time goes by, requirements errors are more expensive to fix

Stage discovered	Relative repair cost (p.day)
Requirements	0.1 – 0.2
Design	0.5
Coding	1
Unit test	2
Acceptance test	5
Maintenance	20

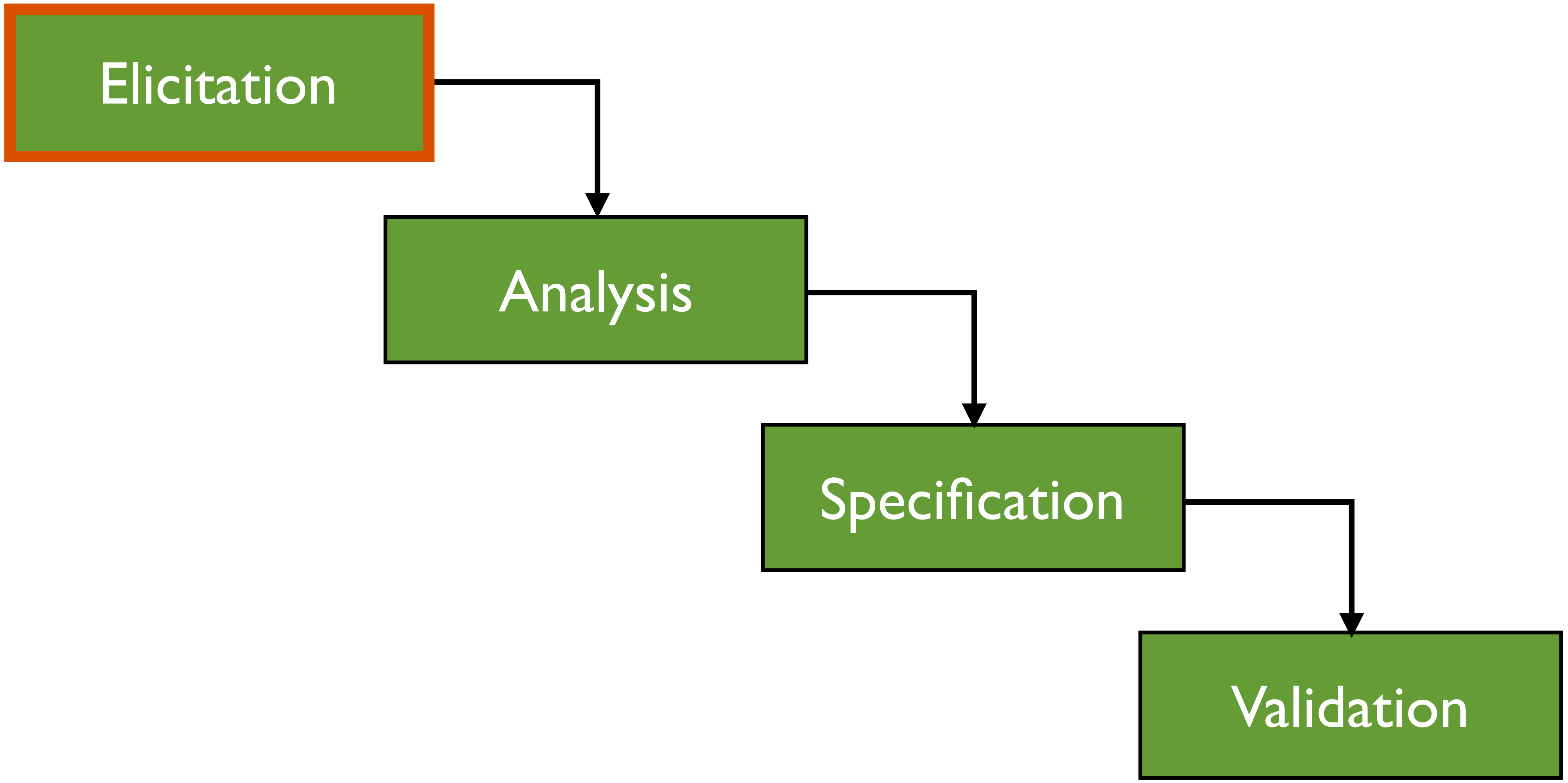
Requirement Engineering Process



Requirements Engineering Process

- Determine the requirements of a system, and specify what behavior is realized
  - Work with stakeholders to elicit the requirements
  - Analyze and model the requirement
  - Document the requirements in a software requirements specification
  - Validate the software specification

Requirement Engineering Process



Requirements Elicitation

- Elicitation is to gather
  - Functions that the system should perform
  - Non-functional requirements that the system should exhibit
- Elicitation is critical but difficult
  - Customers are not good at describing what they want
  - Software engineers are not good at understanding what customers want
  - Customers and software engineers speak different languages

Elicitation Approaches

- Brainstorming
- Interviewing
- Ethnography
- Strawman/Prototype

Brainstorm

Brainstorm

- Gather stakeholders, collect ideas from them and prune ideas
- Organization Tips:
  - Keep the number of participants "reasonable"
  - Invite the most creative people to multiple sessions

Brainstorm - the Storm

- Purpose
  - Generate as many ideas as possible
  - Quantity, not quality, is goal at this stage
- Common Rules
  - No criticism or debate
  - Write down all ideas where all can see
  - No self-censor or wondering if an idea is practical
  - Original list does not get circulated outside of the meeting

Brainstorm – the Calm

- Purpose
  - Review, consolidate, combine, clarify, and expand
  - Rank the list by priority somehow; choose a winner
- Common Rules
  - Go over the list and explain ideas more carefully
  - Categorize into "maybe" and "no" by pre-agreed consensus method (e.g., voting, priority points)
  - Be careful about time: meetings should have a fixed time frame (90-120 minutes)

Brainstorm: Pros & Cons?

- Pros
  - No Preliminary Knowledge Preparation
  - Comprehensive gathering of ideas, solve conflicts earlier
- Cons
  - No clear mission, costly for gathering, may take a long time
  - People from different field may feel difficult to interact

Brainstorm Applicability

vs.

- Good: Startup software, general topic, e.g. personal shopping software
- Not good: Domain experts/systems exist, limited resources, e.g., ATM banking system

Brainstorm: Exercise

- Online bookstore
- Functional requirements
  - e.g., the system shall allow a registered user to log-in to their account
- Non-functional requirements
  - e.g., system login shall take less than 5 seconds.

Interviews

Interviews

- Formal or informal interviews with stakeholders
- Two types of interview
  - Closed interviews: based on pre-determined list of questions
  - Open interviews: issues are explored with stakeholders.
- Effective interviewing
  - Prompt the interviewee to get discussions going using a requirement proposal
  - Be open-minded, avoid pre-conceived ideas

Interviews in practice

- Mostly used approach for requirement elicitation: in almost every project
- Normally a mix of closed and open interviews

Interviews: Pros & Cons

- Pros
  - Simple to apply in practice
  - Usually can get some progress
- Cons
  - Interviewee may ignore details because they are too familiar with them
  - Interviewee may have too little knowledge in computer science to express their ideas effectively

Talk-based Requirement Gathering

- Terminology Problem
- Sometimes people fails to articulate what they want
  - Too familiar to come up with
  - Fail to go to the details

Practice-based requirement gathering

Ethnography

Ethnography: Origin

- A social scientist spends a considerable time observing and analyzing how people actually work.
- People do not have to explain or articulate their work.
- Social and organizational factors of importance may be observed.

Ethnography for Requirement

- Discover requirements by observing the way that people actually work
- This include the interaction and collaboration between people doing their work
  - Talks, emails, meetings, …
- Gathering of real data generated in the actual work
  - reports
  - filled forms
  - emails, …

Ethnography: Pros & Cons?

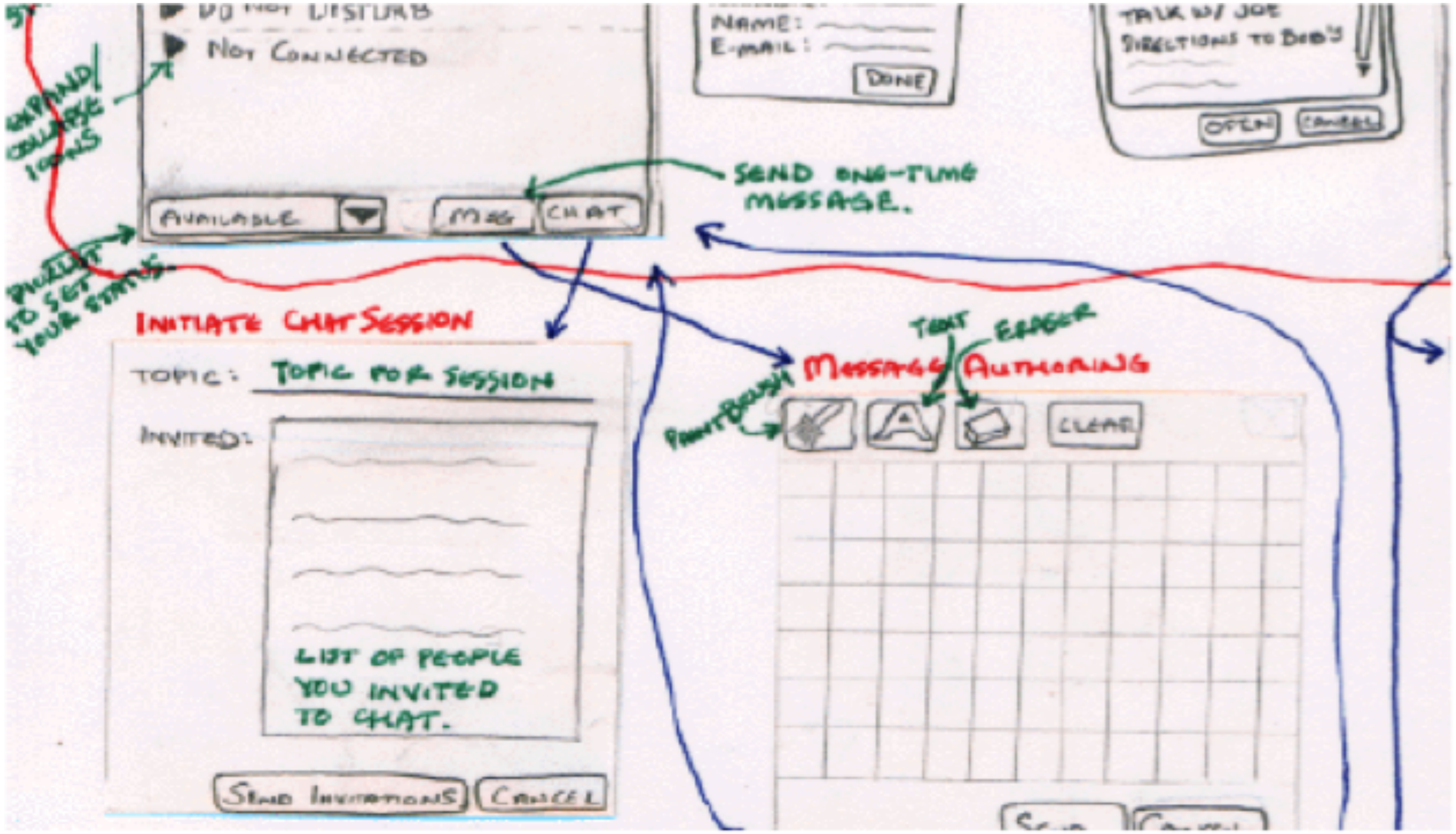
- Pros
  - Reveal real requirements, avoid problems caused by imprecision in oral/written expression
  - Require little preliminary knowledge
- Cons
  - May take a long time to finish an effective observation
  - May have legal or privacy issues
  - Can only observe what is happening at present, but people's behavior may change with the new software
- Frequently used in practice when there is an existing system in use

Strawman/Prototype

Strawman/Prototype

- Prototype: write a prototype software for requirements
- Strawman: something to convey ideas without coding
  - GUI
  - Web pages
  - Flow chart of UIs

Strawman: flow chart



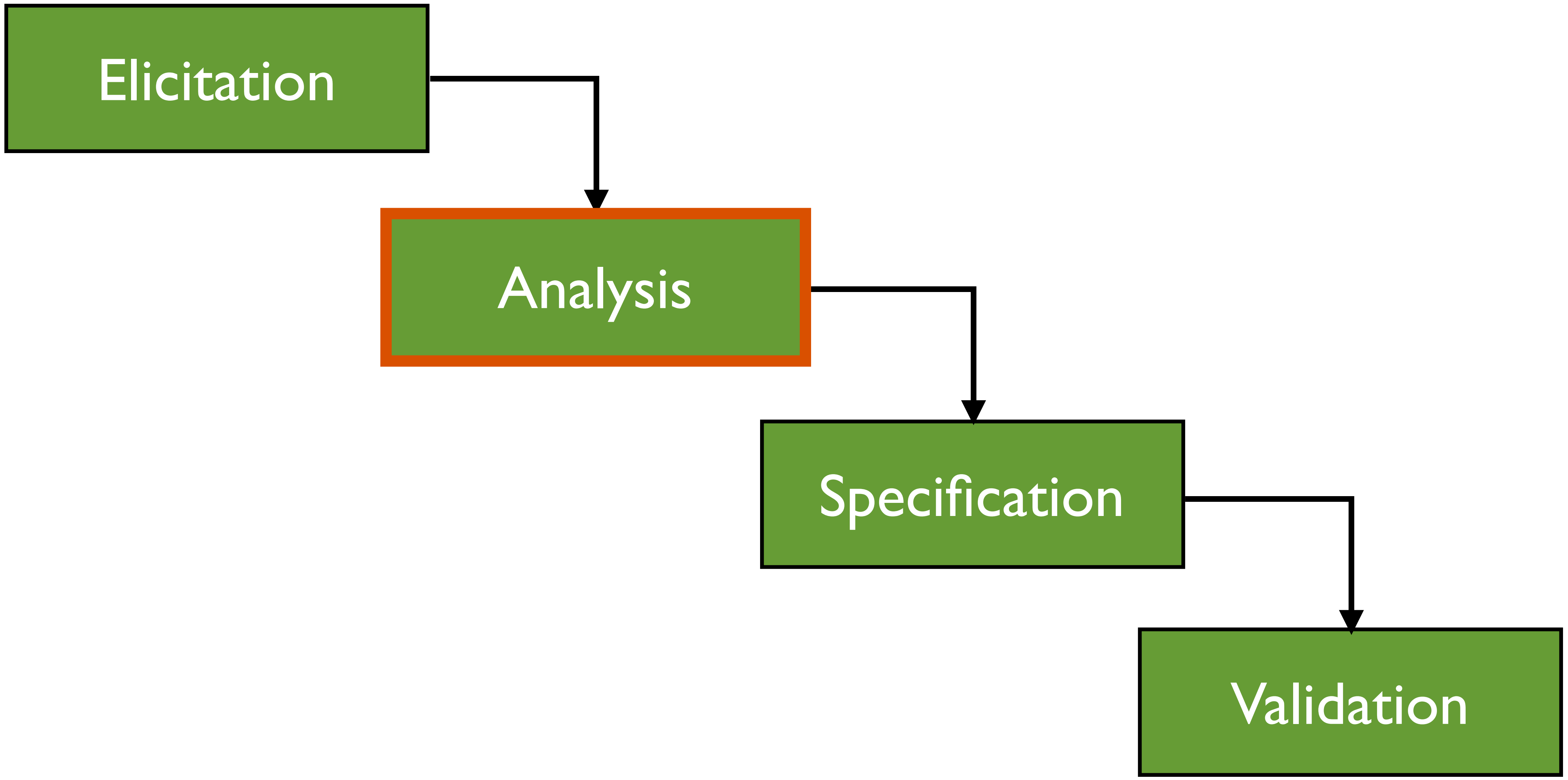
Strawman/Prototype: Pros & Cons

- Pros
  - Can go to details
  - Easy to link requirements to design/implementation
  - Most accurate
- Cons
  - High cost in preparation
  - Require preliminary knowledge
  - Pre-assumptions may limit the scope of the software

Combination of Different Approaches

- Brainstorm + interview
  - Raise some questions, then ask more people
- Interview + strawman/prototype
  - Talk to interviewee with a strawman/prototype
- Interview + ethnography
  - Asking people after observing their work
- Prototype + ethnography
  - Observe how people work on a prototype

Requirement Engineering Process



Requirements Analysis

- Requirements analysts have to understand the system from each stakeholder's point of view
  - Stakeholders have different views of the system
- Requirements analysts resolve conflicting views
- Requirements analysts prioritize requirements
  - Essential requirements
  - Desirable requirements
  - Optional requirements

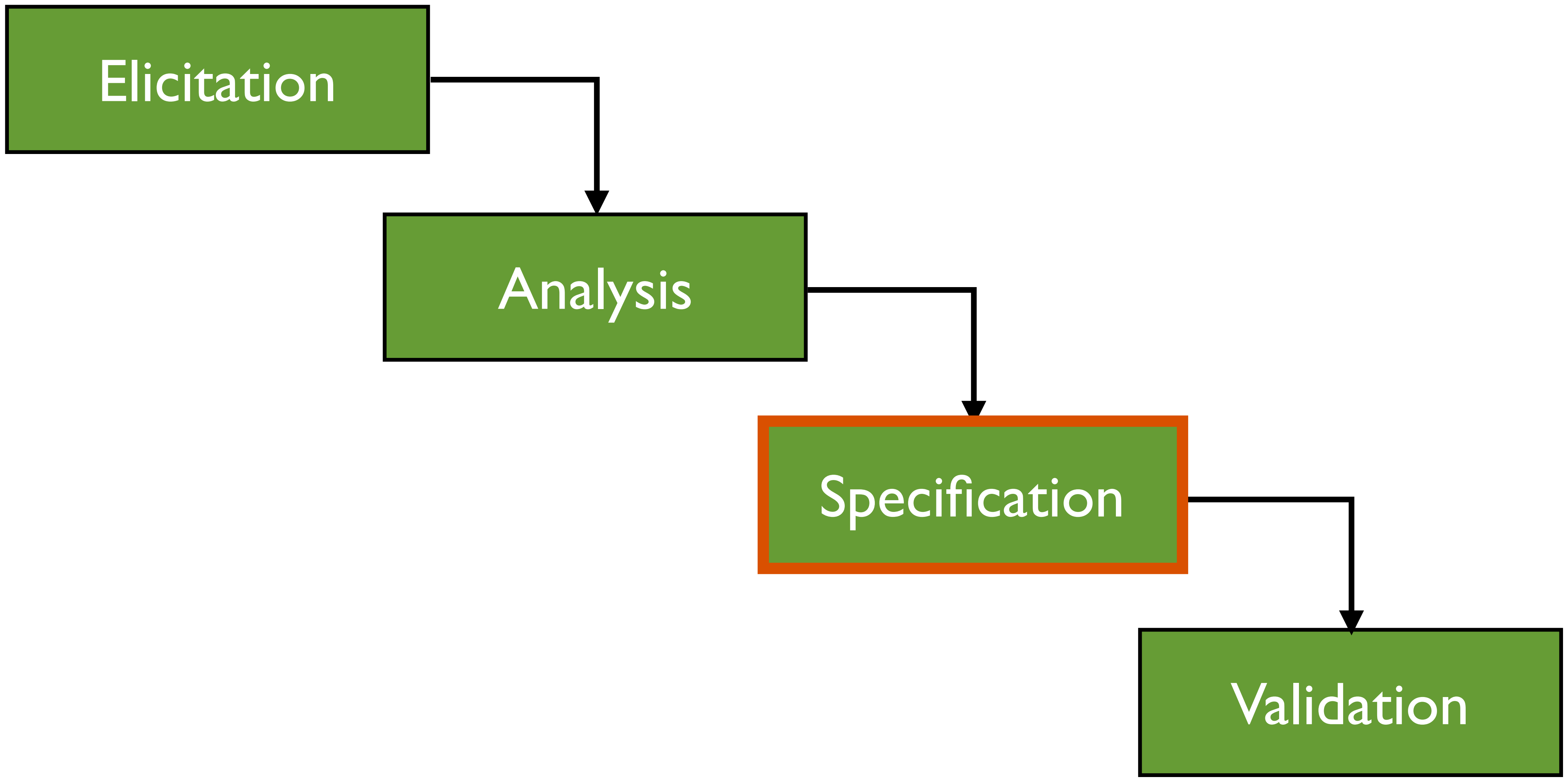
Requirements Analysis

- Goal
  - Determine the scope of the software
- Categorization, negotiation, and decision:
  - Few established fixed approaches
  - Large amount of mental work based on domain knowledge
- Project manager/customer representative often plays the key role

Requirements Analysis: Exercise

- Online bookstore
- Priority
  - Essential
  - Desirable
  - Optional
- Practicality
  - good ideas
  - ideas need further study
  - unusable ideas
- New requirements?

Requirement Engineering Process

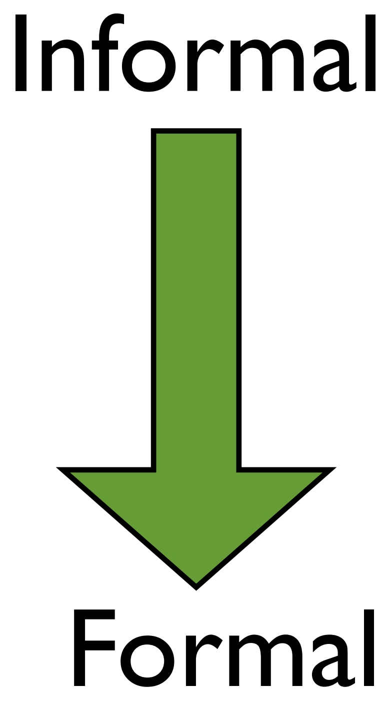


Requirements Specification

- Specify requirements
    - Document what is required for the system
    - State the requirements from the perspective of the developers
    - May be a formal document (IEEE-SRS)

Requirements Specification

- Natural Language Specification
- Structured Specification
- Graph Notation Specification
- Mathematical Specification



Natural language specification

- The requirements are written using numbered sentences in natural language.
- Each sentence should express one requirement.
- Diagrams and tables can be used for better understanding of the specification

Guidelines for writing requirements

- Formatting
  - Invent a standard format and use it for all requirements

    Font, size, indentation, …
  - Use text highlighting to identify key parts of the requirement
- Wording
  - Use language in a consistent way.

    E.g. always use shall for mandatory requirements, should for desirable requirements
  - Avoid the use of computer jargon
  - Include a list of term definitions

Guidelines for writing requirements

- Contents
  - Avoid ambiguity in expression
  - Add as much details as you can (think as a developer)

An example of natural language specification

1.1 If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales for the current month is under 5%

- Any problems with this specification?

Potential term inconsistency: sales & actual sales

An example of natural language specification

1.1 If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales for the current month is under 5%

- Any problems with this specification?

Lack of details:
What are contents in the report?
When and how to print?

An example of natural language specification

1.1 If sales for current month are below target sales, then report is to be printed, unless difference between target sales and actual sales for the current month is under 5%

- Any problems with this specification?

Ambiguity: 5% of actual sales or target sales?

Advantages of Natural Language

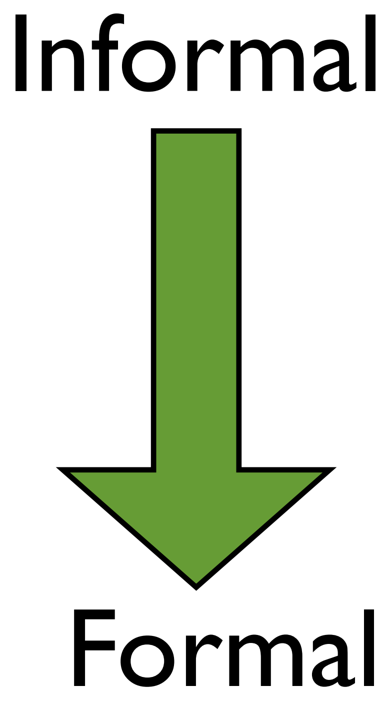
- Expressive, can express almost any concepts, although not precisely
- Can be understood by users, customers, developers, etc.
- Easy to write

Problems with natural language

- Ambiguity, imprecision
- Contradictions can happen
- Several different requirements may be expressed together
  - Functional and non-functional requirements tend to be mixed-up

Requirements Specification

- Natural Language Specification
- Structured Specification
- Graph Notation Specification
- Mathematical Specification



Structured specifications

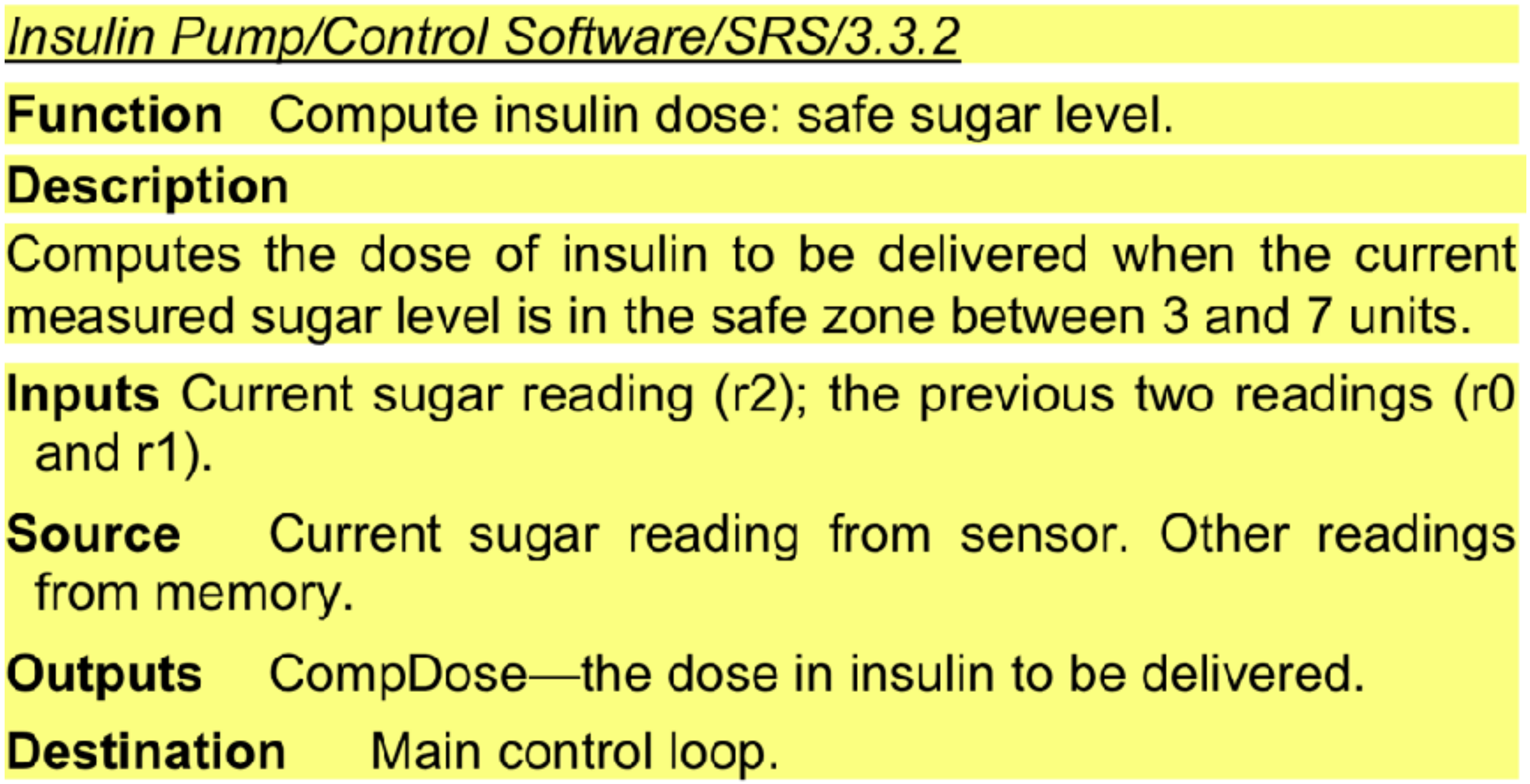
- The structure of a requirement is predefined
- The freedom of the requirements writer is limited
- Some common structures:
    - Forms
    - Tables

Form-based specifications

- Definition
  - Definition of the function or entity
  - Description of the action to be taken
- Input & Output
  - Description of inputs and where they come from.
  - Description of outputs and where they go to
  - Pre and post conditions (if any)
- Dependencies
  - Information needed & other entities used
  - The side effects (if any) of the function

    E.g., reduced credit score when you query it

Example: Insulin pump for blood sugar control



Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

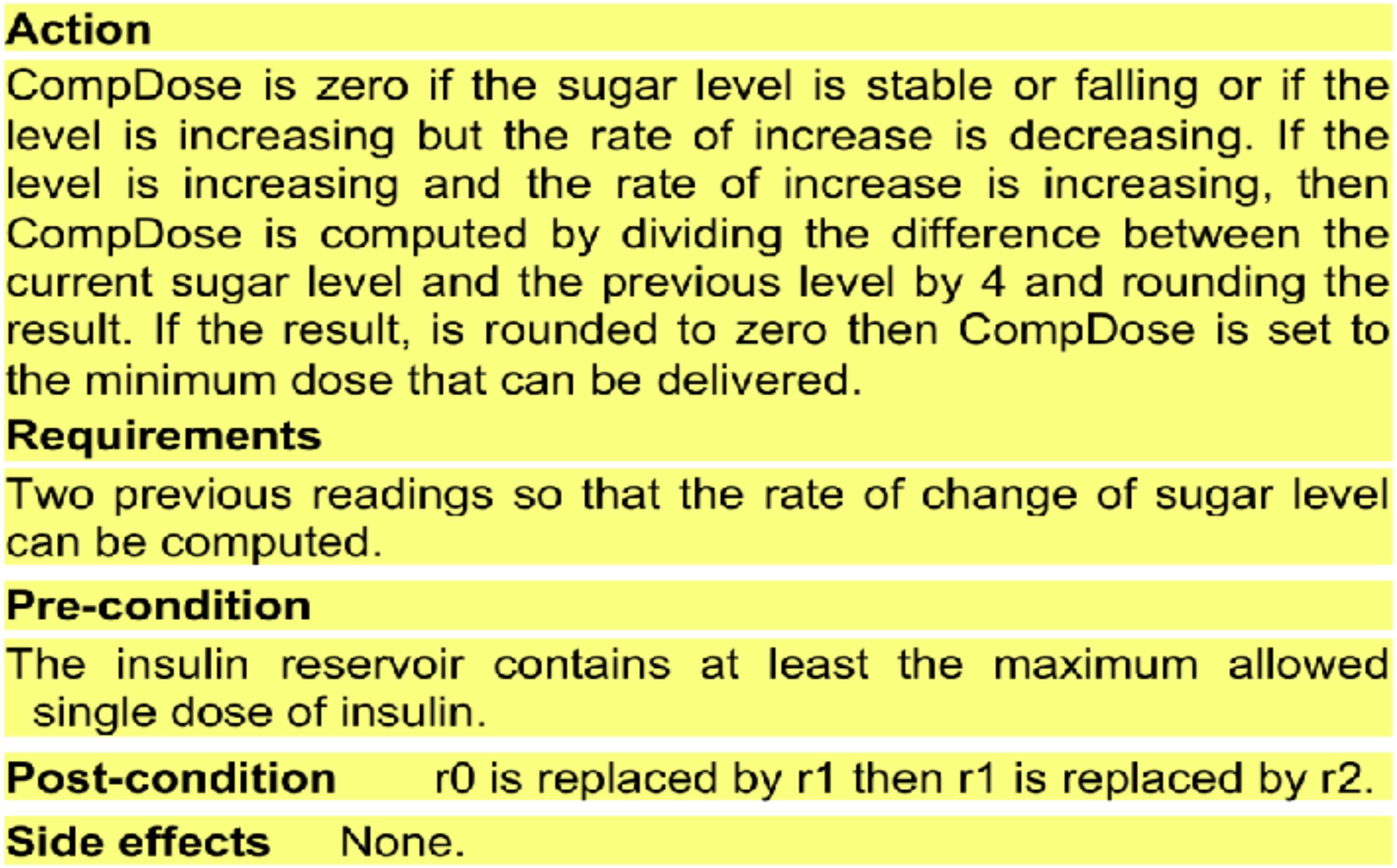
Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

Example: Insulin pump (Cont'd)



Action	
CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.	
Requirements	
Two previous readings so that the rate of change of sugar level can be computed.	
Pre-condition	
The insulin reservoir contains at least the maximum allowed single dose of insulin.	
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

UT DALLAS

Tabular specification

- A map from inputs to outputs in the form of a table
  - Each line corresponds to a case in inputs
  - The corresponding action is filled in after inputs
- Particularly useful when you have to define a number of possible alternative actions

Example: Insulin Pump

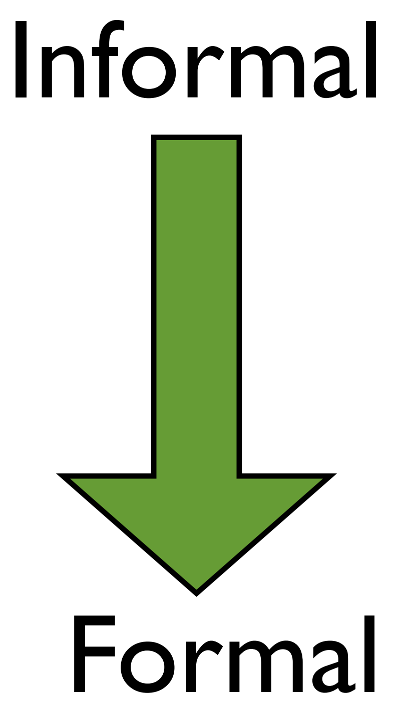
Condition	Action
Sugar level falling ($r2 < r1$)	CompDose = 0
Sugar level stable ($r2 = r1$)	CompDose = 0
Sugar level increasing and rate of increase decreasing $((r2 - r1) < (r1 - r0))$	CompDose = 0
Sugar level increasing and rate of increase stable or increasing $((r2 - r1) \geq (r1 - r0))$	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

Pros & Cons: structures

- Pros
  - Easier to control quality compared with pure natural language
  - Still easy to write and understand
  - Reduce imprecision and missing of details
- Cons
  - The form of structure has strong impact on the quality of specification, and is not easy to design
  - Less expressiveness due to the rigid structures
  - Still has the problem of natural language expression, such as ambiguity, missing term definitions, etc.

Requirements Specification

- Natural Language Specification
- Structured Specification
- Graph Notation Specification
- Mathematical Specification

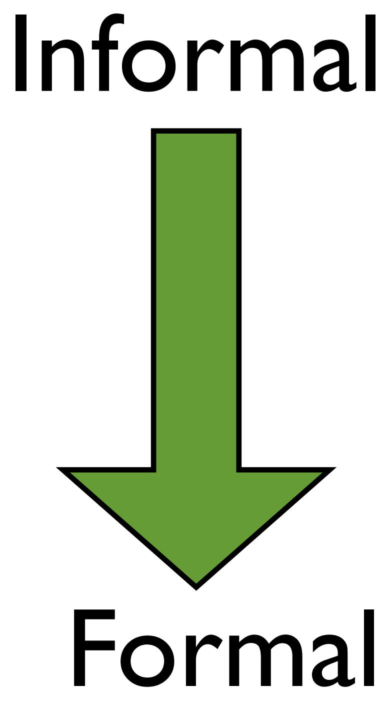


Graph Notation Specification

- Predefined Graphical models
- Supplemented by text annotations
- Existing techniques:
  - UML: Use-case diagram

Requirements Specification

- Natural Language Specification
- Structured Specification
- Graph Notation Specification
- Mathematical Specification



Mathematic Specification

- Write specification using predefined formal languages
- Define all concepts, inputs, and corresponding outputs /actions formally
- Some popular specification languages:
  - Z language
  - Alloy
  - …

Introduction to Z

- Z "notation" was first developed at the Programming Research Group at Oxford University
- Z is a "specification" language based on sets, relations, and functions to express:
  - What are the functionalities of the system
  - And what the desired results are
  - But without stating the "how" part
- Thus Z is a declarative language - - meaning that it is non-imperative (not like Java or C)

Declarative vs. Imperative

- Imperative language: telling the "machine" how to do something, and as a result what you want to happen will happen
  - e.g., C, Java, C#,...

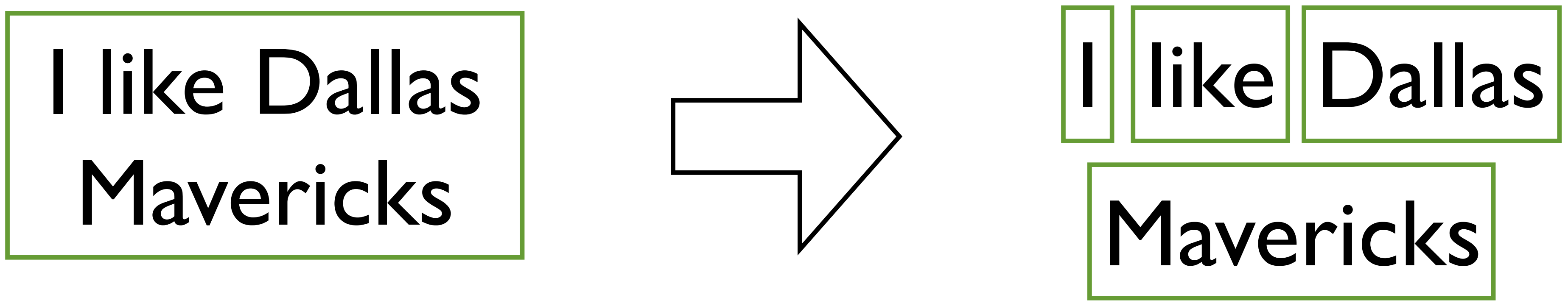
```
for(Element e in list) {
    if(e.size <10) { new_list.append(e);}
}
```

- Declarative language: telling the "machine" what you would like to happen, and let the computer figure out how to do it
  - e.g., Z, Alloy, SQL…

```
SELECT size FROM table WHERE size < 10;
```

Example: Word Split with Z

- Textual Description
  - Purpose: Breaking a text into words
  - A text is a sequence of characters.
  - Blank characters: spaces, line breaks, and tabs
  - A word is a sequence of non-blank characters
  - A separator is a sequence of blank characters.



Example: word split with Z language

- Concept Definition:

```
char == [CHARACTER]
blank == [space, line break, tab]
TEXT == seq char
SEPARATOR == seq1 blank
WORD == seq1 (char \ blank)
```

CHARACTER is defined as all possible characters

seq denotes a sequence of elements from its set-type argument

seq1 denotes a sequence of **non-empty** elements from its set-type argument

“\” denotes exclude

Example: word split with Z language (Cont'd)

- Requirement of function *words*:

```
words: TEXT -> seq WORD
\forall s: SEPARATOR; w: WORD; l,r: TEXT
   words  <> = <> &
   words  s = <> &
   words  w = < w > &
   words (sr) = words r &
   words (ls) = words l &
   words (lsr) = words l + words r
```

Pros & Cons

- Pros
  - Precise, little ambiguity (almost pseudo code)
  - Computer readable, so correctness can be checked with automatic tools (e.g. model checker)
  - Easy to write test case based on the specification (providing oracles)
- Cons
  - Hard to understand
  - Hard to write, costly to find people writing it and using it
  - Expressiveness depending on the specification language (often not general enough)

In practice

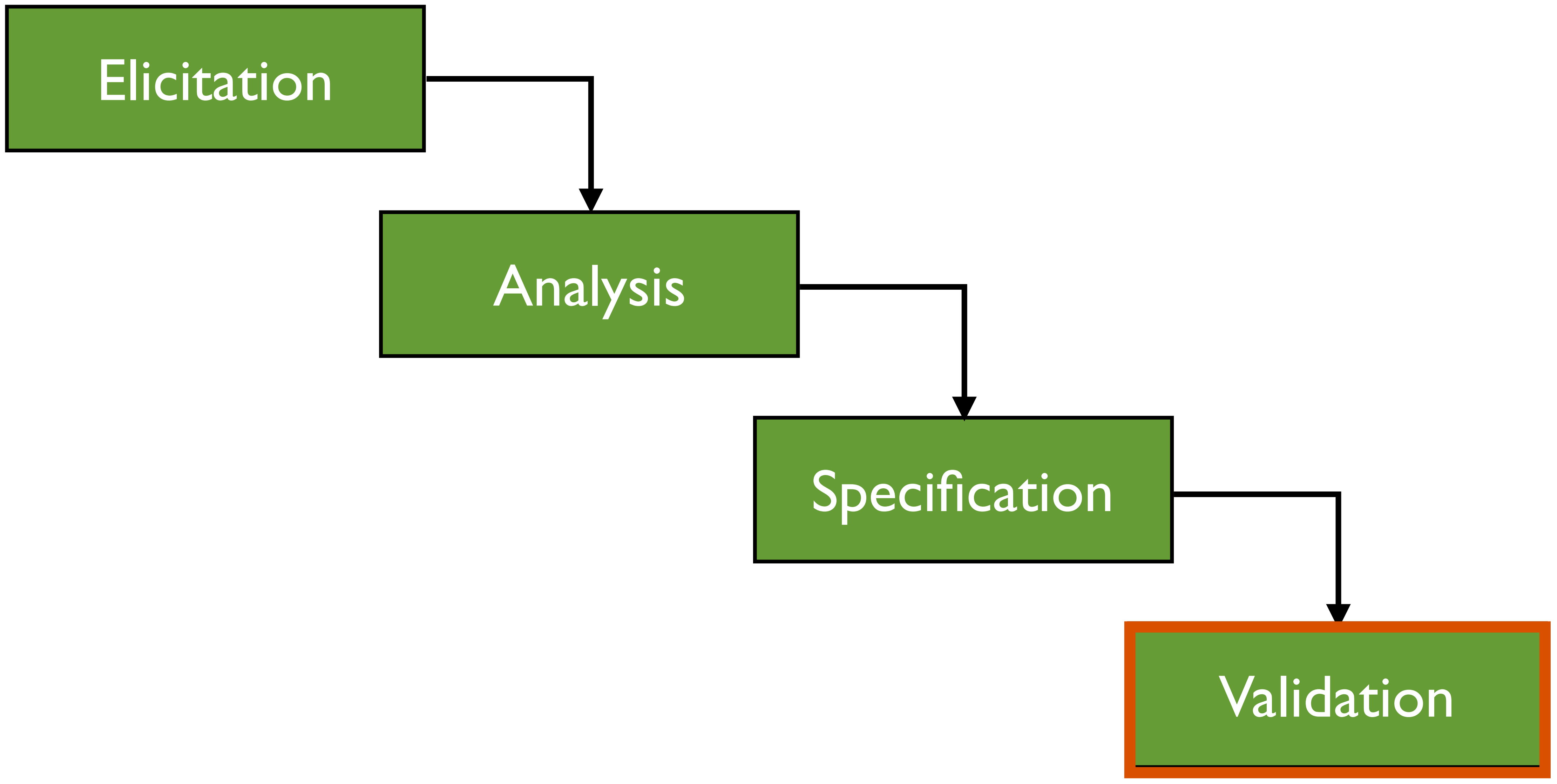
- Natural language
  - Widely used, especially for small projects
- Structure
  - Often used as a supplement to natural language
- Graph notation
  - Widely used in industry, business information systems
- Mathematics
  - What software often involves mathematical specification?

    Compilers (programming language)

    Browsers (HTML)

    Database systems (SQL)

Requirement Engineering Process



Requirements Validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
    - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error

Requirements Validation (Cont'd)

- Validation can be done with techniques
  - Review
  - Prototype
  - Writing test cases
  - Verification of specifications

Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements
  - Regular reviews should be held while the requirements definition is being formulated
  - Both client and contractor staff should be involved in reviews
  - Reviews may be formal (with completed documents) or informal

Requirements validation techniques

- Prototyping
  - Using an executable model of the system to check requirements.
  - Which software process model use this technique?
    Prototype model
- Test-case generation
  - Developing tests for requirements to check testability
  - Which software process model use tests to validate requirements?
    Extreme programming

Specification Verification

- Verification can be done with techniques
  - Consistency checking

    No contradictions
  - Completeness checking

    All concepts are well defined
  - Formal verification of the above or other properties

    Usually require mathematical specification

    Model checking, automatic reasoning, …
- Which software process model often use specification verification?

    Waterfall model