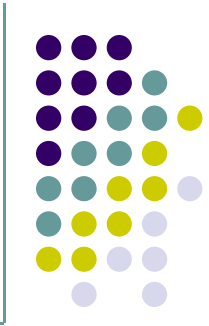
Professor Hausi A. Müller PhD PEng FCAE
Department of Computer Science
Faculty of Engineering
University of Victoria

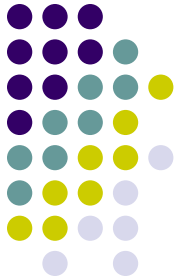
http://www.engr.uvic.ca/~seng321/
https://courses1.csc.uvic.ca/courses/201/spring/seng/321

Announcements



- Feb 1-5
  - I will be in Los Angeles running meetings as Vice President of IEEE Computer Society
  - Tue/Wed, Prof. Kevin Ryan
  - Fri, Lorena Castaneda
  - Labs: UML tutorial
- Feb 8-13
  - Reading Break
  - No classes
  - No labs
- Deliverables
  - Post your deliverables on your web site; do not submit electronically
  - C0 due today
- Textbook Reading Assignment
  - Chapter 16—Lift controller
  - Chapters 1-3 Elicitation
  - Chapters 8-10 Elicitation and Modelling
- Midterm
  - Fri, Feb 26 in class
  - 3 mid questions today

SENG 321 Calendar



Deliverable S0 due	Fri, Jan 22	S0 Related work	5% of project
Team website due	Tue, Jan 26	Website	5% of project
Deliverable C0	Fri, Jan 29	C0 RFP2 informal req spec	5% of project
Reading break	Feb, 8-12	No class	No labs
Deliverable S1 due	Tue, Feb 16	S1 formal req spec	10% of project
Deliverable C1	Thu, Feb 18	C1 feedback on S1	5% of project
Midterm	Fri, Feb 26	In class	14% of project

IEEE computer society

- The IEEE Computer Society (CS) annually sponsors over 200 geographically diverse technical conferences, symposiums, and workshops dedicated to providing computing professionals with innovative forums designed to facilitate the identification, creation, capture and exchange of highly peer-reviewed scientific and technological knowledge that benefits members, the profession, and humanity.

- Prof. Hausi Müller, who serves on IEEE CS Board of Governors, is the *2016 Vice President of Technical & Conference Activities*

University of Victoria Engineering

Join IEEE as a Student Member

Join IEEE

Why join IEEE?

When you join IEEE, you join a community of over 425,000 technology and engineering professionals united by a common desire to continuously learn, interact, collaborate, and innovate. IEEE Membership provides you with the resources and opportunities you need to keep on top of changes in technology; get involved in standards development; network with other professionals in your local area or within a specific technical interest; mentor the next generation of engineers and technologists, and so much more.

On this Page:

- Benefits
- Qualifications and dues
- Membership and testimonial videos
- Additional resources
- Current members

Join as a professional Join as a student

www.ieee.org/membership_services/membership/join/index.html

University of Victoria Engineering

Next Week (Tue/Wed)

Professor Kevin Ryan
University of Limerick, Ireland

- Lero
  - http://www.lero.ie/
  - Director Emeritus of Irish Software Centre (Lero)
- Research area
  - Requirements engineering
  - A cost-value approach for prioritizing requirements

Where Do Requirements Come From?

- Requirements come from users and stakeholders who have demands and needs
- A requirements analyst or engineer
  - Elicits demands and needs (raw requirements)
  - Analyzes them for consistency, feasibility, and completeness
  - Formulates them as requirements and writes down a specification for each requirement
  - Validates that the gathered requirements reflect the needs and demands of stakeholders:
    - Yes, this is what I am looking for ☺
    - This system will solve my problems ☺

Questions that Arise During Requirement Gathering

- Is this a need or a requirement?
- Is this a nice-to-have vs. must-have?
- Is this the goal of the system or a contractual requirement?
- Is this a legal requirement?
- Is this a green requirement?
- Do we have to program in Java? Why?
- Do we have to leverage certain middleware?

Types of Requirements

- Data/Functional:
  - What is the system supposed to do?
  - Mapping from input to output
- Non-functional/Quality:
  - Process: time to market, standards, delivery, modifiability, extensibility, portability, adaptability
  - Product: usability, usefulness, efficiency, reliability, availability, resiliency, self-management
  - External: cost
- Context / environment:
  - Range of conditions in which the system should operate

Project Inception

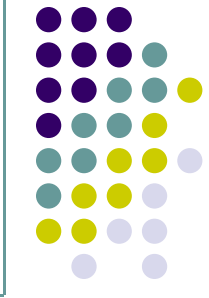
- Reasons to start a project:
  - Immediate need: Too many customers, we need a better system to manage our customer relations
  - Competitive pressure: We are losing customers because of our current web search engine
  - Bright idea: A portable mp3 player
  - Seeing what others have: Web browser is pretty useful; we should have our own. GUI on Mac vs. Windows
  - Long term nuisance: When our server crashes we lose data not flushed to disk (Journal based file systems); refactoring, reengineering

Issues to Resolve during Project Inception

- Issues to be addressed early in a project:
  - **Goals:** What are benefits of proposed system? Why would someone buy it?
  - **Scope:** What is included/excluded?
  - **Scope:** What would the final system look like? Imagine we are celebrating the project's success – what would we celebrate? How do you recognize success?
  - **Cost/Benefit:** Will the system succeed? Can we make money on it? Who will (tangible/intangible) benefit?
  - **Stakeholders:** Who cares? Who is affected by system?

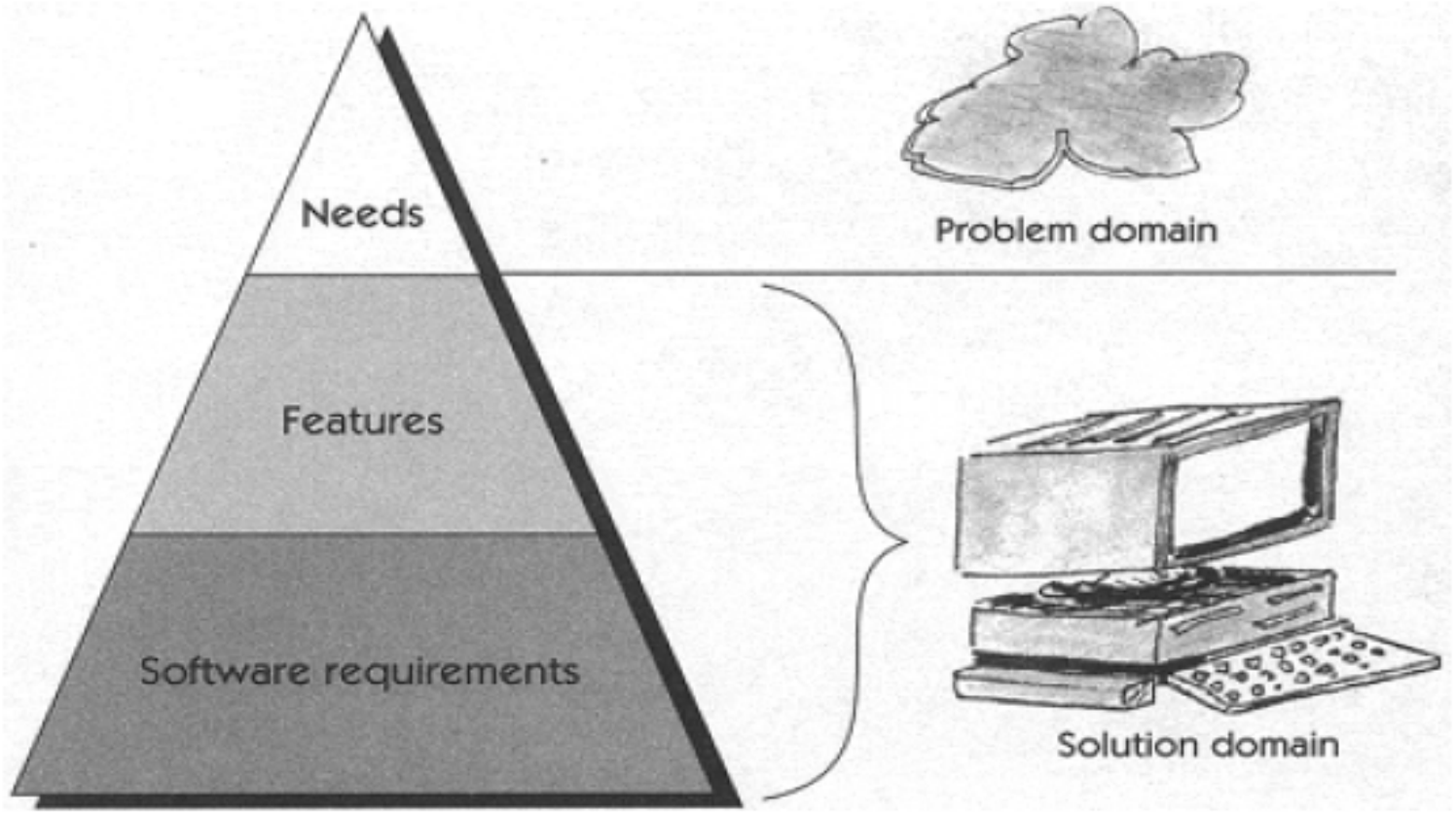
- **Goals and Scope are the first version of the raw requirements**

Tangible and Intangible Benefits



- Tangible benefits
  - Benefits that can be measured in terms of money
- Intangible benefits
  - Subjective benefits that cannot be measured in monetary terms

Stakeholder Needs, Desired Features, System Requirements

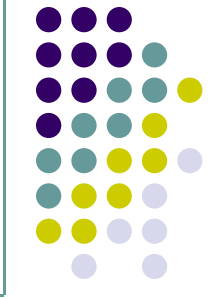


Ambiguity

- The user can enter a name. It can be 127 characters:
  - Must the user enter a name?
  - Can the name be less than or greater than 127 chars?
- The system should prominently display a warning message whenever a user enters invalid data:
  - What does *should* mean?
  - What does *prominently* mean?
  - Is *invalid data* defined?

[Cohn 2004]

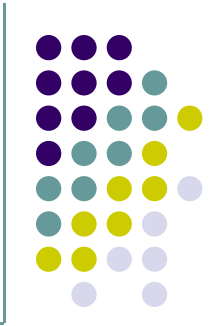
Examples of Ambiguity



- Entrée comes with soup or salad and bread:
  - (Soup or Salad) and Bread
  - (Soup) or (Salad and Bread)
- A panda walks into a restaurant …
  - Eats, shoots, and leaves
  - Eats shoots, and leaves

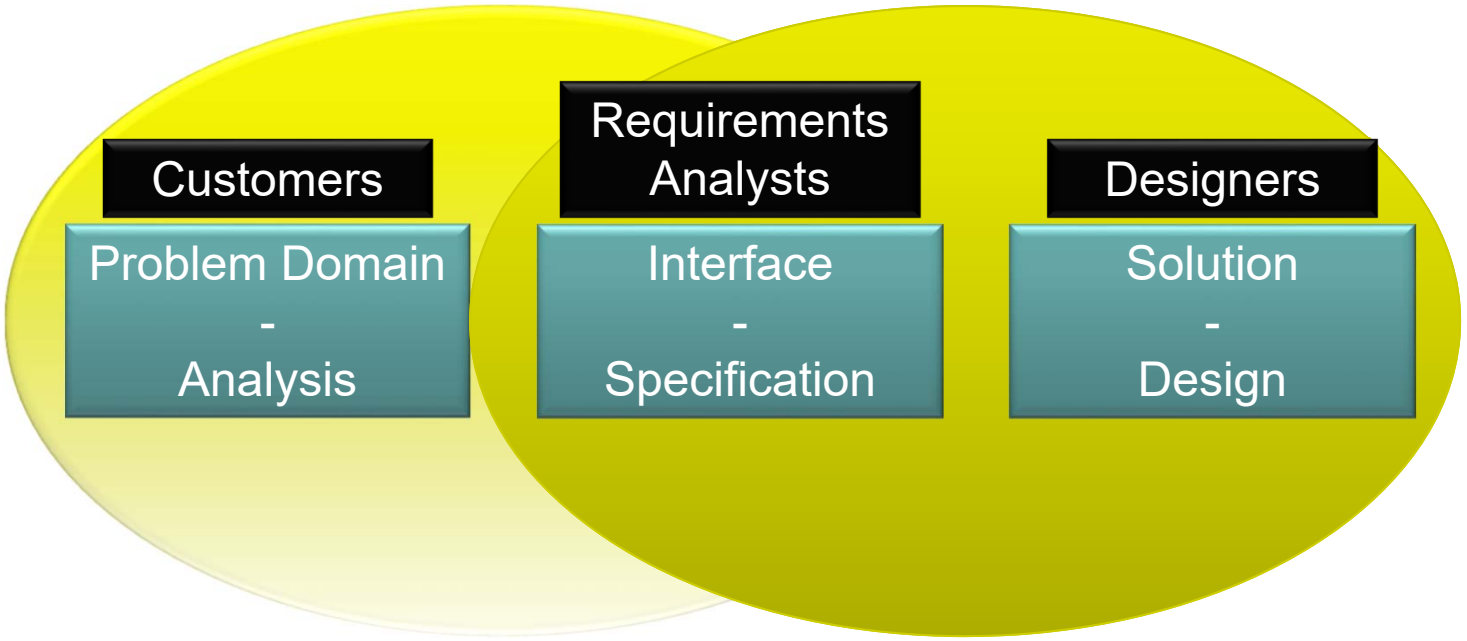
[Cohn 2004]

Ariane 5 Disaster

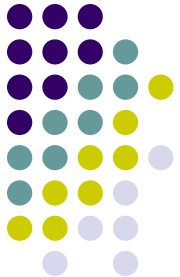


- Flight 501 (June 4, 1996) was the first, and unsuccessful, test flight of the European Ariane 5 expendable launch system
  - Due to a malfunction in the control software, the rocket veered off its flight path 37 seconds after launch and was destroyed by its self-destruct system.
  - The breakup caused the loss of four Cluster mission spacecraft, resulting in a loss of more than US$370 million.
- One of the most infamous computer bugs in history
  - Ariane 5 reused the specifications from the Ariane 4, but Ariane 5's flight path was different and beyond the range for which the reused program had been designed.
  - Due to the different flight path, a data conversion from a 64-bit floating point to 16-bit signed integer value caused an arithmetic overflow (i.e., the floating point number had a value too large to be represented by a 16-bit signed integer).
  - Efficiency considerations had led to the disabling of the exception handler (Ada code).
  - This led to a cascade of problems, culminating in destruction of the entire flight.
- http://en.wikipedia.org/wiki/Ariane_5_Flight_501

Separation of Concerns

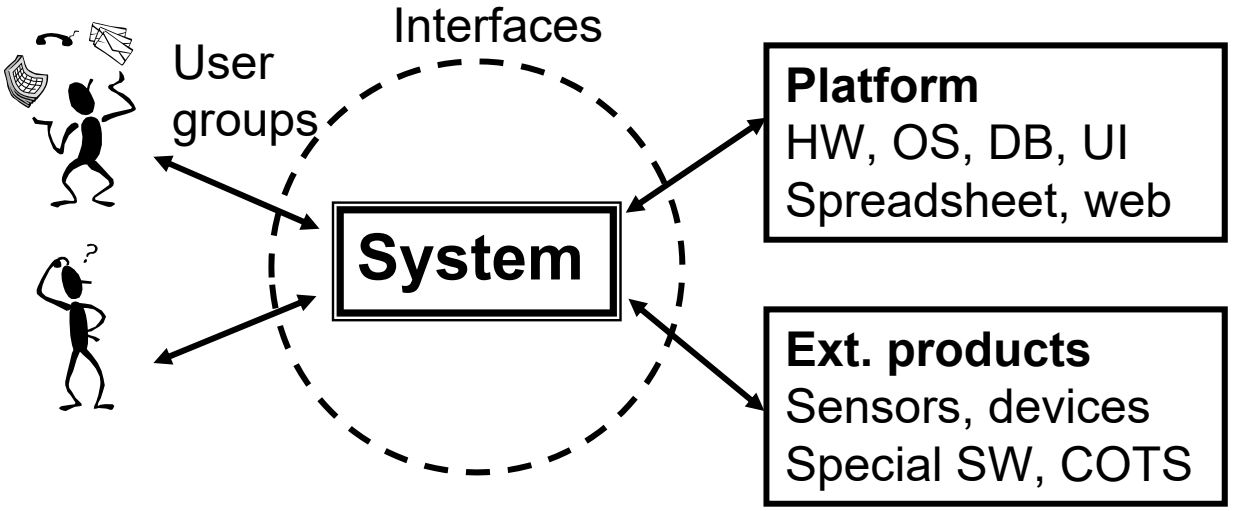


eDesign vs. iDesign



- External design versus internal design
- External design
  - Critical part of requirements engineering
  - System's appearance
  - System's behaviour
  - In other fields called styling and design
- Internal design
  - Not part of requirements engineering
  - Decomposing system into parts
  - Architecture
  - How does the system actually work
  - How is the system implemented
  - In other fields called engineering

Contents of a Requirements Specification



Data requirements

System state: Database, files, communication states, input/output formats

Functional requirements, each interface

Record, compute, transform, transmit
Theory: F(input, state) -> (output, state)
Function list, pseudo-code, activity diagram
Screen prototype, support tasks xx to yy

Quality requirements

Performance
Usability
Maintainability …

Other deliverables

Documentation
Install, convert, train …

Managerial requirements

Delivery time
Legal
Development process …

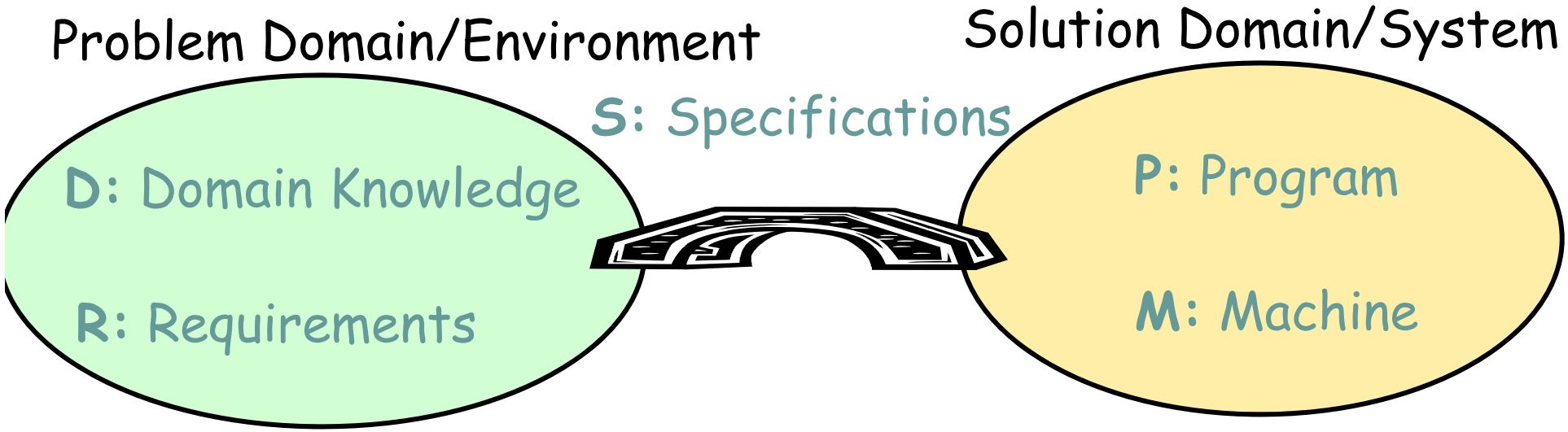
Helping the reader

Business goals
Definitions
Diagrams …

Problem Data vs. Solution Data

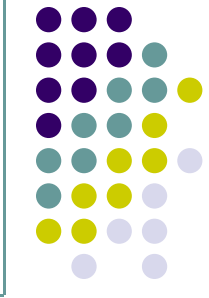
- Problem data
  - Originate in the problem domain
  - Stored and transformed by the solution system
  - Subject to requirements analysis
  - Should be fully defined in requirements documents
- Solution data
  - Originate in the solution domain
  - Input and output to drive the solution
  - Intermediate data

Software Requirements and Specifications



- **D:** Relevant Facts about the environment that are assumed true whether or not we ever build the system
- **R:** Things that we wish to be made true by delivering the proposed system
- **S:** A description of the behaviours the program must have in order to meet these requirements **R**

Relevant Facts about the Environment



- If assumption of truth does not work then system will fail
- Write everything that is relevant even “obvious” facts that may change
- The system will behave as desired only if these assumptions are not violated
  - No concurrent updates, at most 100 users
- Raw requirements
  - Exist *outside* of the system you are building.
  - Have nothing to do with data structures, algorithms, GUIs, 32 bit **`int`**s, databases, or any other internal part of the system (except shared phenomena).

System Failures

- System will behave as desired only if assumptions about the environment are not violated:
  - What if customers share PINs?
  - What if plane skids on runway? Wheels would not turn...
  - What if users jump over the turnstile?

Example 1
Account Access

- **Environment**
  - Domain Knowledge: Customers have PINs, PINs are not shared
  - Requirement: Account accessed only by authorized person
- **System**
  - Program: Internal security and encryption algorithms
  - Machine: Keypad and screen provided
- **Specification**
  - Users must *input* the correct PIN to gain access to an account

Example 2
Avionics Software

- **Environment**
  - Domain Knowledge: wheel pulses on iff wheel is turning, wheel is turning iff plane is on runway
  - Requirement: Reverse thrust only enabled when the aircraft is moving on the runway
- **System**
  - Program: Internal algorithms and data structures to monitor and analyze the wheel pulses data
  - Machine: Wheel pulses, Reverse thrust mechanism
- **Specification**
  - Reverse thrust enabled iff wheel pulses is on
- **Problem**
  - Airplane “aquaplaned” on the runway one wet day.
  - The wheels didn’t turn, so there were no pulses, so the avionics systems didn’t let the reverse thrust engage.

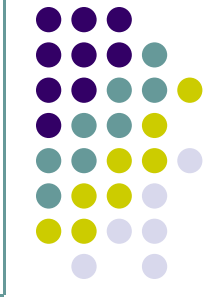
Example 3
Park Entrance Software

- **Environment**
  - Domain Knowledge: User must pass through single entrance to enter park
  - Requirement: User must pay to get into the park
- **System**
  - Program: Internal algorithms and data structures to monitor the coin drop and move turnstile arm
  - Machine: coin slot, turnstile arm
- **Specification**
  - Turnstile arm moves once coin is put in slot

Example 4
Heart Monitoring System

- **Environment**
  - Domain Knowledge: Patient's heart beat, Nurse keep an eye on patient. Heart stopping is a concern
  - Requirement: Nurse must be informed if heart stops
- **System**
  - Program: Internal monitoring algorithms and data structures
  - Machine: Sensor to monitor heart beat, buzzer to warn nurse
- **Specification**
  - System must monitor heart through *sensors* and alert nurse through *buzzer* if heart stops

Requirements: *What* vs. *How*?



- Requirements do not specify **How**!
  - **What:** Account accessed only by authorized person
    - **How:** PIN, password, retina scan, fingerprint
  - **What:** Reverse thrust only enabled when the aircraft is moving on the runway
    - **How:** Altitude measure, wheel pulses, pilot control
  - **What:** User must pay to get into the park
    - **How:** Manned booth, coin/card-operated turnstile
  - **What:** Nurse must be informed if heart stops
    - **How:** audible, visual, buzzer

Requirements: *What* vs. *How*?

- Requirements do not specify **How**!

Key Findings from Last Lecture

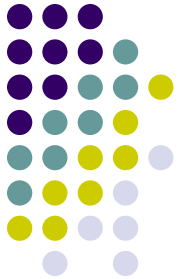
- During requirements elicitation and analysis carefully separate the following concerns
  - eDesign and iDesign
  - Problem data and solution data
  - *What* and *How*
- Requirements do not specify *How*
- Always ask *Why* instead of *When or How*?
- Do not restrict design with unnecessary requirements ➔ unwanted solutions
- For a functional requirement, describe the requirement but not how it is done
- Requirement types: functional and non-functional

Always Ask Why?!

- Ask “Why” instead of “When or how would you do that”?
- Ignorance of the domain is usually preferred
  - The designated ignoramus usually asks questions whose answers are “obvious” to experts, but often expose hidden knowledge that might otherwise not be modeled explicitly.

Professor Dan Berry, Waterloo suggests that one day, requirements engineers will advertise their areas of ignorance (rather than expertise) to get jobs ☺

Ask Why



Neural diagnostics

System shall have mini keyboard with start/stop button, . . .

Why?

Possible to operate it with “left hand”.

Why?

Both hands must be at the patient.

Why?

Electrodes, bandages, painful . . .

Recommendation: Why & How

Measuring neural response is a bit painful to the patient. Electrodes must be kept in place . . .
So both hands should be at the patient during a measurement.

R1: It shall be possible to perform the commands *start, stop, . . .*
***. . .* with both hands at the patient.**

Might be done with mini keyboard (wrist keys), foot pedal, voice recognition, etc.

Domain
- why

Requirements

Example
- how

Do Not Constrain Range of Solutions with Unnecessary Design Requirements

- Requirements with unnecessary design can restrict the range of solutions or result in bad solutions.
- Example:
  - Need capability to drink hot liquids and analyst writes:
    - The thing shall contain up to 8 ounces of hot liquid.
    - *The thing shall have a handle.*
  - In response to a requirements quality review analyst writes:
    - *The thing shall allow a person to hold it without getting burned.*
  - This requirement allowed multiple solutions including
    - a cup with handle, carton sleeve, a Styrofoam cup, or a thermos mug