Electrical & Computer ENGINEERING

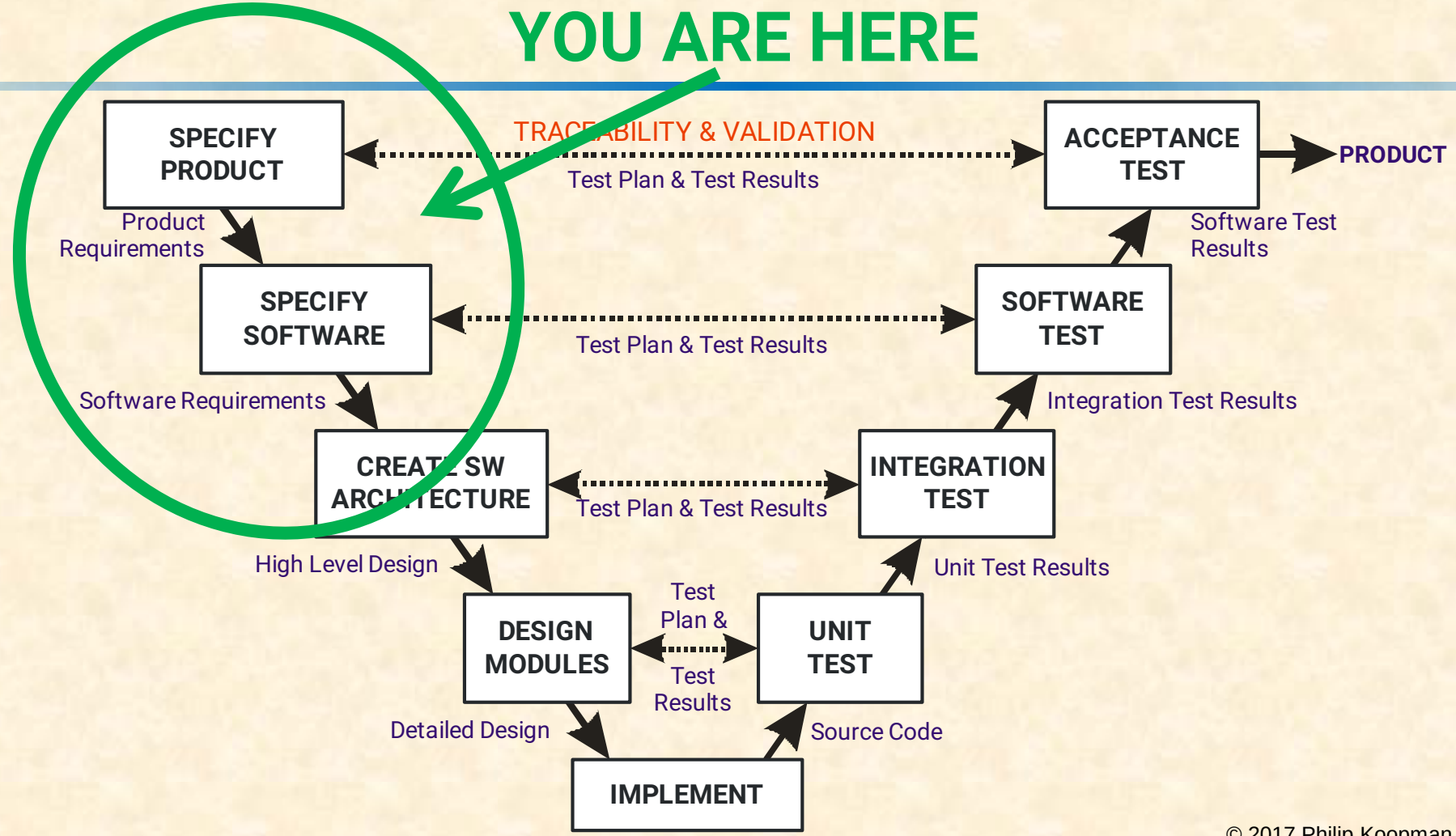
18-642:
Requirements

2/12/2018

"In spite of appearances, people seldom know what they want until you give them what they ask for. "
- Gerald M. Weinberg - Donald Gause and Gerald Weinberg, Are Your Lights On?:
https://goo.gl/bxoBrr

Carnegie Mellon University

YOU ARE HERE



Requirements Overview

- **Anti-Patterns:**
  - **Requirements aren't written down**
  - **Requirements incomplete, imprecise**
  - **"Be like last version, except…"**

Ariane 5 Flight 501

https://www.ima.umn.edu/~arnold/disasters/ariane.html

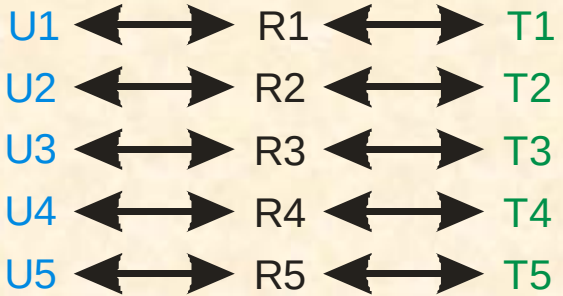
- **Requirements**
  - Requirements faults can defeat a design before it is even built
  - Describe what system does
    - Also what it's not supposed to do
  - Precise, testable language
    - Each requirement traces to system test

- **Technical cause:**
  - Float ➔ 16 int16 horizontal velocity conversion
  - Overflow exception assumed to be HW fault
  - Secondary shut down; primary shut down too
- **Underlying cause was component reuse:**
  - INS reused from Ariane 4 with lower h_velocity
  - Function not even required in flight for Ariane 5!

[http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html]

Rules for Good Requirements

- **Precise and minimally constrained**
  - Describes what system should do, not how it does it
  - Uses "**shall**" to require an action; "**should**" to state a goal
  - If possible has a numeric target instead of qualitative term
    - Has tolerance (e.g., 500 msec +/- 10%, "less than X")
- **Traceable & testable**
  - Each requirement has a unique label (e.g., "R-7.3")
  - Each requirement cleanly traces to an acceptance test
  - Requirement satisfaction has a feasible yes/no test
- **Supported within context of system**
  - Supported by rationale or commentary
  - Uses consistent terminology
  - Any conflicting requirements resolved or prioritized

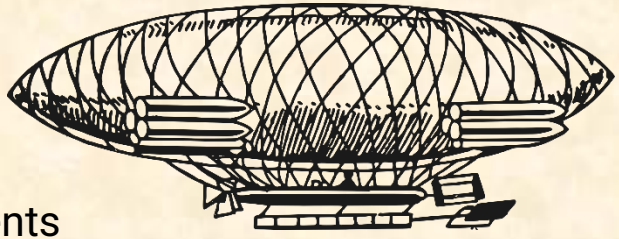


Examples of Problematic Requirements

- **Untraceable** (no label)
  - System shall shut down when E-STOP is activated.
- **Untestable**
  - R-1.1: System shall never crash
- **Imprecise**
  - R-1.7: The system provides quick feedback to the user.
- **No measurement tolerance**
  - R-2.3: LED shall flash with a period of 500 msec
- **Overly complex**
  - R-7.3: Pressing the red button shall activate Widget X, while pressing the blue button should cause LED Z to blink instead of LED Y illuminating steadily, which would be accomplished via the yellow button.
- **Describes implementation**
  - R-8.3: Pressing button W shall cause two 16-bit integer values to be added, then …

Extra-Functional Requirements

- **Emergent properties (things hard to attribute to one component)**
  - Performance, real-time deadlines
  - Security, Safety, Dependability in general
  - Size, Weight and Power consumption ("SWaP")
    - Often handled with an allocation budget across components
  - Forbidden behaviors ("shall not do X")
    - Often in context of safety requirements
    - "Safety function" is a way to ensure a negative behavior, but some behaviors are emergent

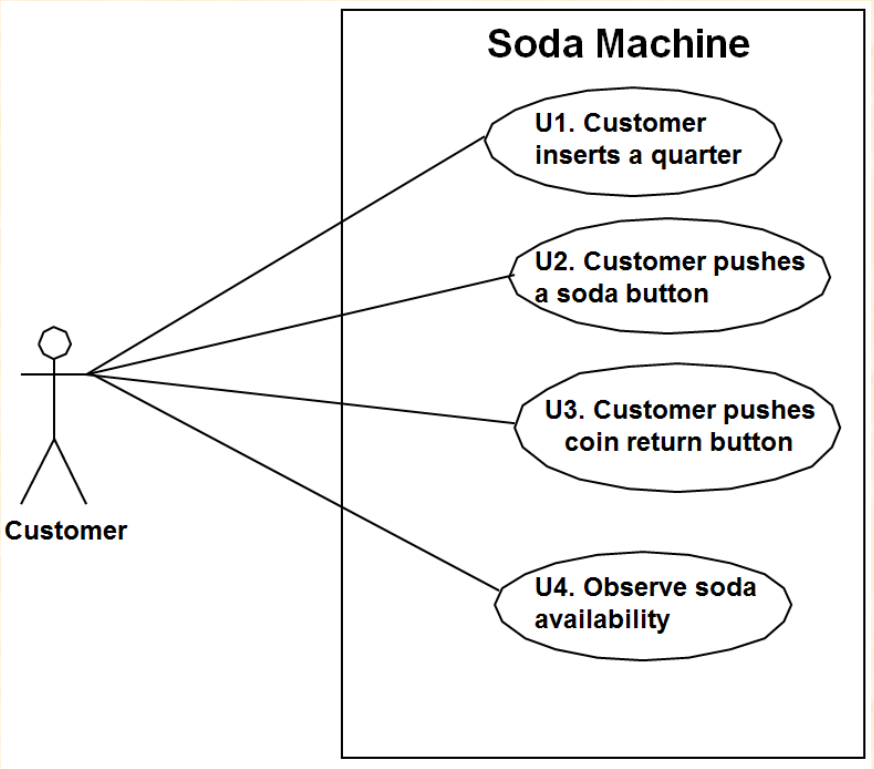


- **Design constraints**
  - Must meet a particular set of standards
  - Must use a particular technology
  - System cost, project deadline, project staffing

https://goo.gl/hT3nDU

Requirements Approaches

- **Text document with list of requirements**
  - Works best if domain experts already know reqts.
  - Over time, this can converge to OK reqts.
- **UML Use Cases**
  - Different activities performed by actors
  - Requirements are scenarios attached to each use case
- **Agile User Stories**
  - Each story describes a system interaction
- **Prototyping**
  - Customers know it when they see it
  - Sometimes a paper mock-up is enough
- **Functional decomposition**
  - Start with primary system functions
  - Make more and more detailed lists of sub-functions (creates a “functional architecture”)



UML Use Case Diagram

Example Software Requirements

National Aeronautics and Space Administration

Parent Req	ReqID	Requirement Text and Rationale	Prior -ity	Allocated To
FSRD-3714	SRS-3.2.6.12	The Software **shall** send data at a user data rate from zero up to and including 100 Mbps. *Rationale: The maximum data rate the Payload Avionics Software must send is 100 Mbps. Lower rates must also be handled.*	P2	PAS
FSRD-3133	SRS-3.2.6.13	The software **shall** send and receive data on two SpaceWire channels simultaneously at up to the maximum SDR interface data rate (full duplex) that can be sustained by both SDRs. *Rationale: When communicating with multiple radios, the Software will need to sustain an achievable data rate. In this requirement, it is defined as the minimum data rate of the two (or three, if possible) SDRs involved in the experiment. For instance, this data could be provided in the routing table. If two other radios are involved, then the data rate may change, based on the capability of those two radios (i.e. a new minimum interface data rate). This value should not be hard coded, but should have the capability for change, once on-orbit.*	P2	PAS

GRC-CONN-REQ-0084
EFFECTIVE DATE: 02/23/2010

https://goo.gl/qct5tL

Best Practices for Requirements

- **Six C-terms for Good Requirements**
  - Clear, Concise, Correct, Coherent, Complete and Confirmable
- **Also:**
  - Deal with extra-functional issues
  - Relate requirements to design flow
    - E.g., associate with user stories or use cases
- **Requirements pitfalls**
  - Avoid unnecessary details and implementation
  - If it's missing from requirements, it won't get done
  - If it's not testable, you won't know if it got done

https://goo.gl/6H3dxi