Trustworthy Systems Through Quantitative Software Engineering

Larry Bernstein
Stevens Institute of Technology
Castle Point, Hoboken, NJ 07030
USA

Trustworthy Software is:

- Safe: Does no harm
- Reliable: No crash or hang.
- Secure: No Hacking Possible

Come Fly with Me

Characteristics

- Quantitative Specifications
- Designed for Trustworthiness
- Bounded Execution Domains
- Certified against Requirements
- Certified against Problem
- Reliability Tested
- Stress Tested
- Diabolically Tested
- Defined Development Process

The Airbus A320

Background

- First civilian fly-by-wire computer system so advanced can land plane virtually unassisted
- No instrument dials – 6 CRTs

What Happened?

- In three crashes, the pilots claim the plane was higher than CRT indicates.
- Altitude read 67ft before the wheels had even left the ground!
- The fly-by-wire system could ignore pilot actions.

Poor Designs in A320

- Programmed landing maneuvers with bug in altitude calculation
- Warning system alerts only seconds before accident; no time to react
- Flight path angle and vertical speed indicator have the same display format; confuses pilots.

Airbus 340 Sept 1994 UK

- Software bug computes wrong fuel level.
- Unhelpful Help, “Please wait ...”
- Plane turns right when told to turn left.
- Plane drops at 9 degrees elevation when told to drop at 3 degrees.

Untrustworthy Software

- Buggy software
- Pilots either frantic or bored.
- Error and warning messages are often numerous or indecipherable, so pilots ignore them.

No worries, I just drove a Saturn from Short Hills on Rt. 206!

Software Hazard Analysis

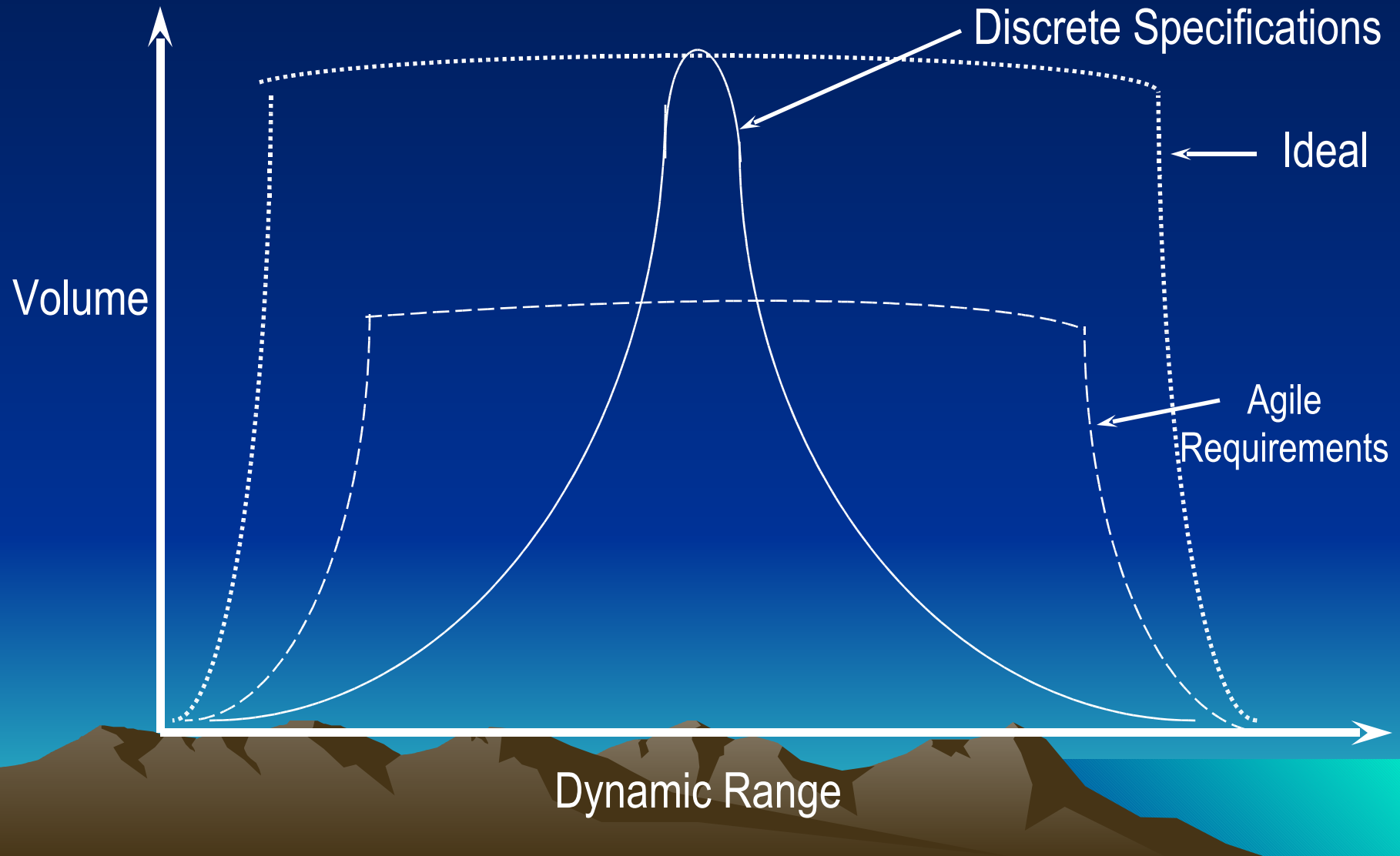
- Upfront Safety Plan
- Detailed Analysis Reports
- Risk Identification
- Risk Assessment
- Continue analysis throughout development and system life.

NIST Special Publication 500-223

What is a Requirement?

- A property that must be exhibited by a system to solve some problem.
- Requirements may be
  - Functional providing product capabilities
  - Non-functional constraining the implementation
- Trustworthy requirements are non-functional

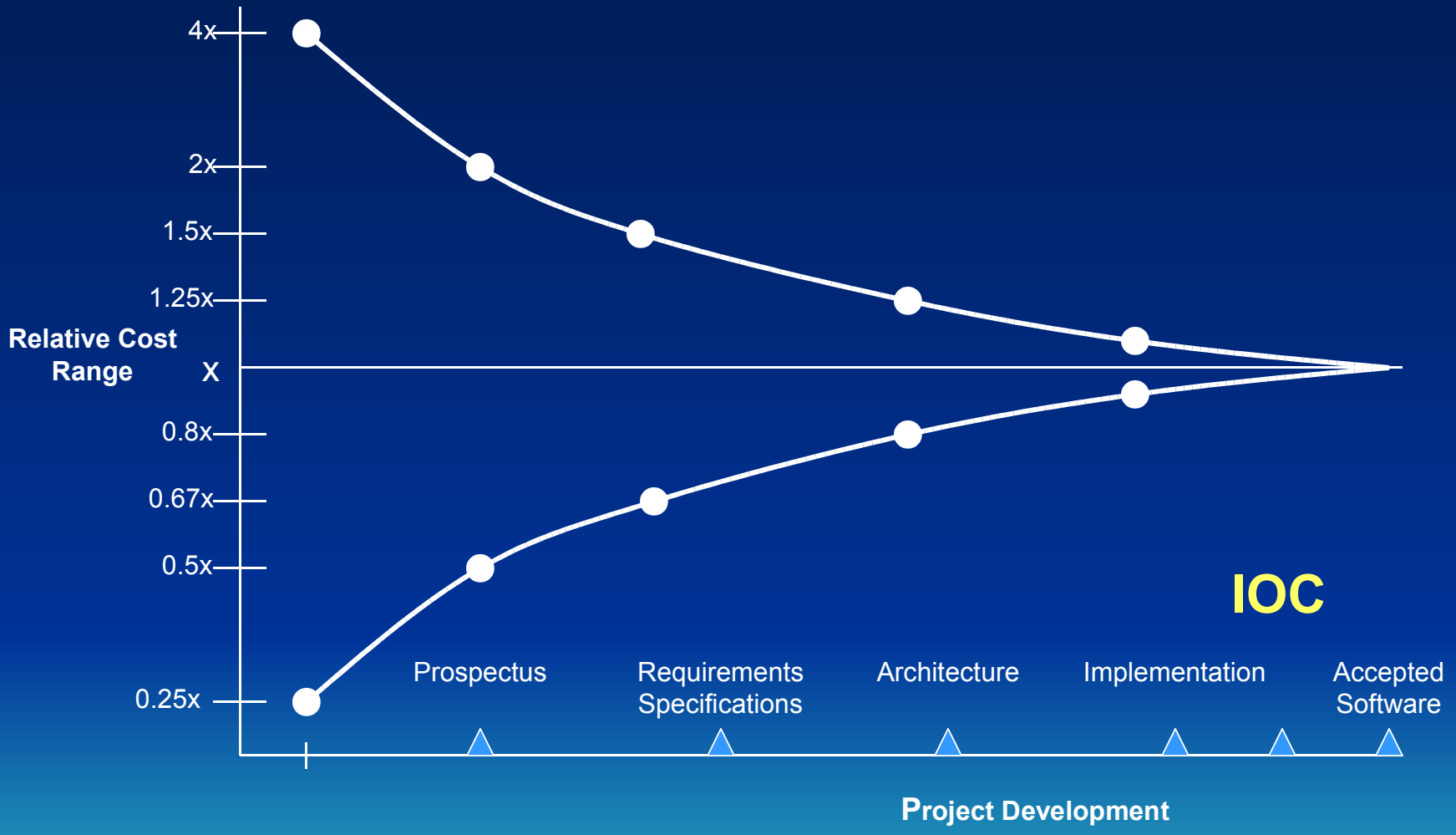
System Performance Resulting from Robust Requirements vs. Discrete Specifications



Top Ten Software Risk Items

Category	Risk Item
People	1. Personnel Shortfalls
	2. Unrealistic Schedules and Budgets
Requirements	3. Developing the Wrong Software Functions
	4. Developing the Wrong User Interface
	5. Gold Plating
	6. Continuing Stream of Requirements Changes
Externalities	7. Shortfalls in Externally-Furnished Component
	8. Shortfalls in Externally-Performed Tasks
Technology	9. Real-Time Performance Shortfalls
	10. Straining Computer Science Capabilities

Relative Project Costs



Highlights of Quantitative Approach

- Lambda Protocol
- Overlaps with Systems Engineering
- Industrial Strength Requirements for Software Intensive Systems-of-Systems

Universal Software Engineering Equation

$$Reliability(t) = e^{-k\lambda t}$$

when the error rate is constant and where k is a normalizing constant for your software shop and

λ = Complexity/ [effectiveness x staffing]

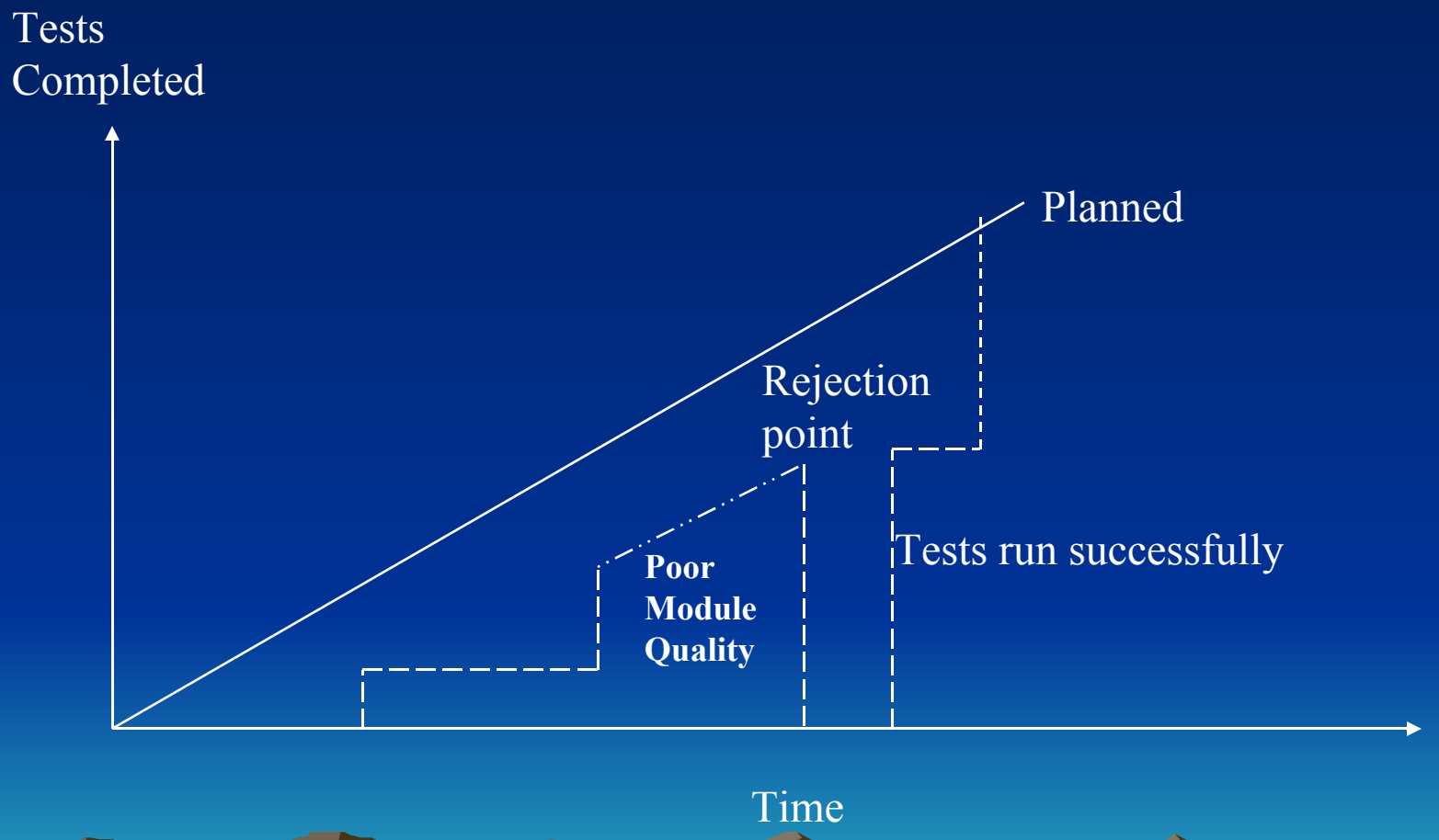
Boundary Conditions

$$\text{Reliability}(0) = 1$$

$$\text{Reliability}(T) = e^{-k\lambda T}$$

$$\text{Reliability}(\infty) = 0$$

Software Testing Footprint



QSE Characteristics

- Solving the right problem the right way
- Tested against requirements.
- Certified against problem
- Bounded execution domain

QSE Lambda Protocol

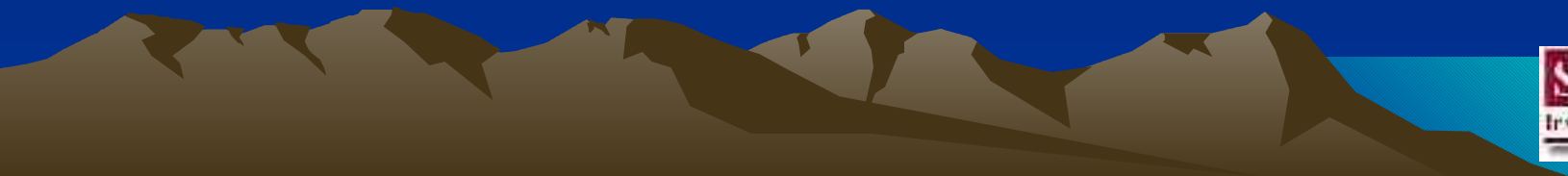
- Prospectus
- Measurable Operational Value
- Prototyping or Modeling
- sQFD
- Schedule, Staffing, Quality Estimates
- ICED-T
- Trade-off Analysis

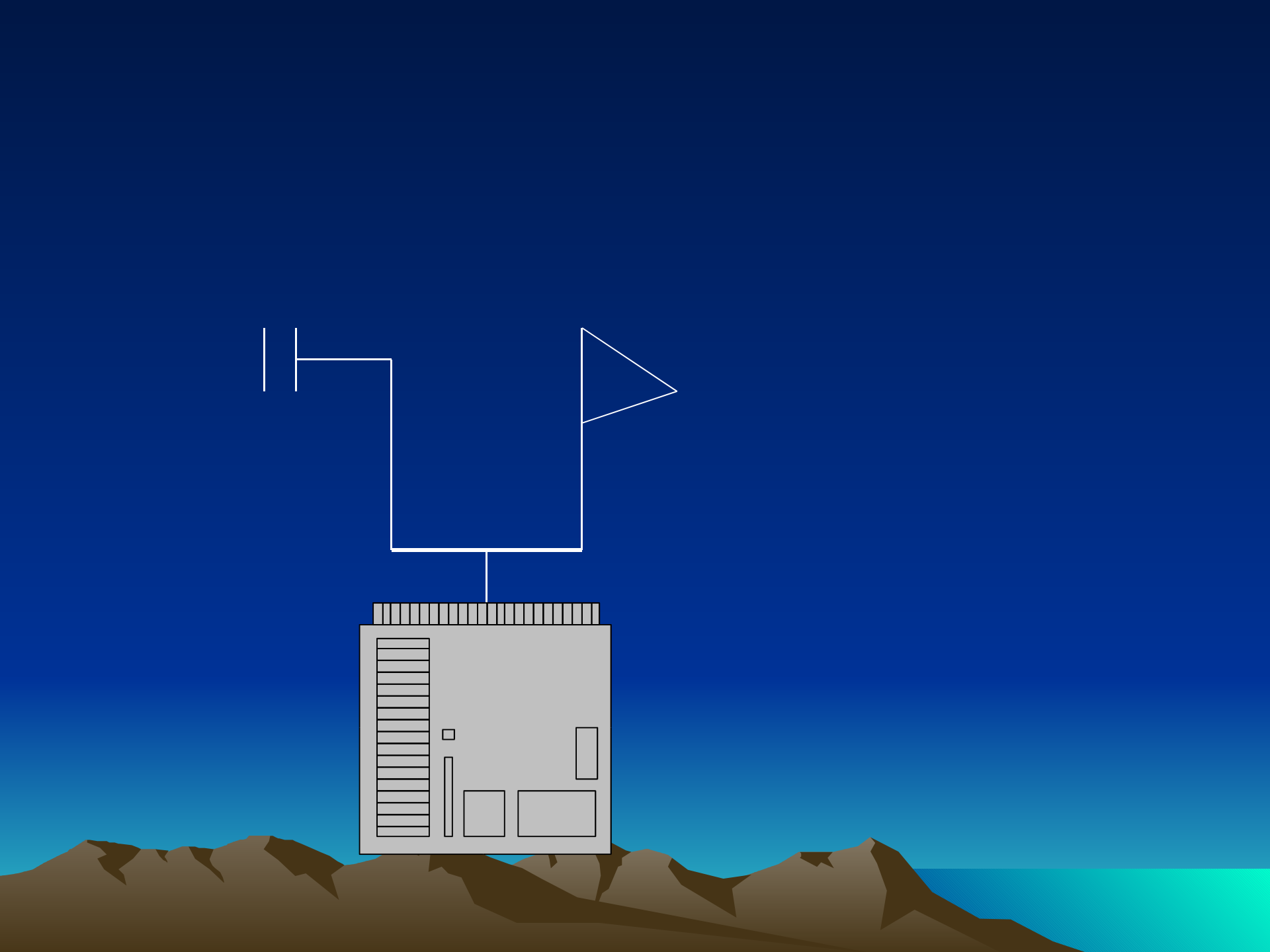
Prospectus

- Description of the problem domain
- Scope of solution
- Specific project goals
- Constraints on the behavior or structure of the software:
  - For example, Trustworthiness

Case study: the Mars Explorer

Bugs on Mars



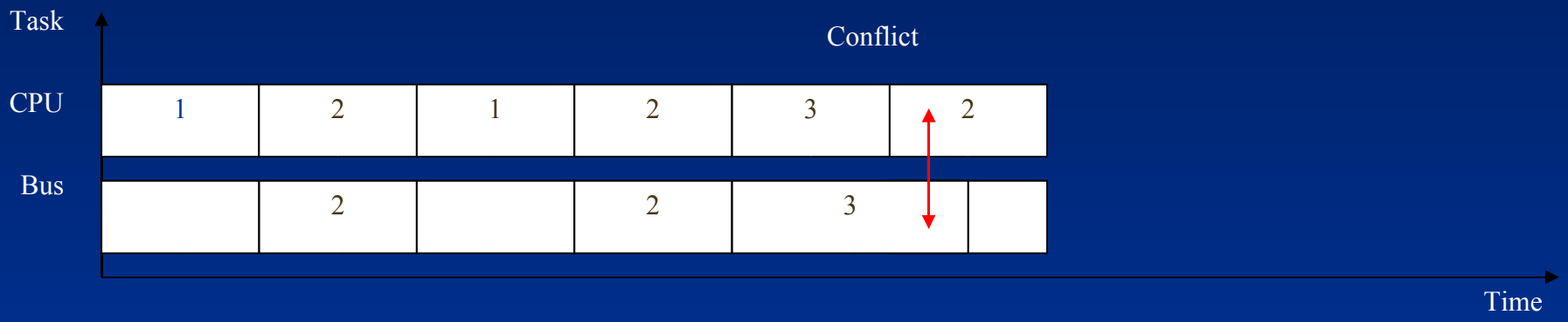


There was a mismatch between priorities set in:

- The extended machine
- The software application
- The software that controls the bus

Priorities were

Task 1 Reboot
Task 2 Gather data
Task 3 Send images



Deadlock

Priorities Should Be

Task 1 Reboot
Task 3 Send images
Task 2 Gather data

Rejuvenation

- Restart the process at fixed time periods to purge aging effects.
- Run a program one day 365 times, not a year.

Periodic preemptive rollback prevents future failures.
Gracefully terminating an application allows restarting at a fixed internal state.

Conditions That Cause Unreliability

- Poor Algorithms
- Missing Deadlines
- Roundoff Error Build Up
- Memory Leaks
- Broken Pointers
- Register Misuse

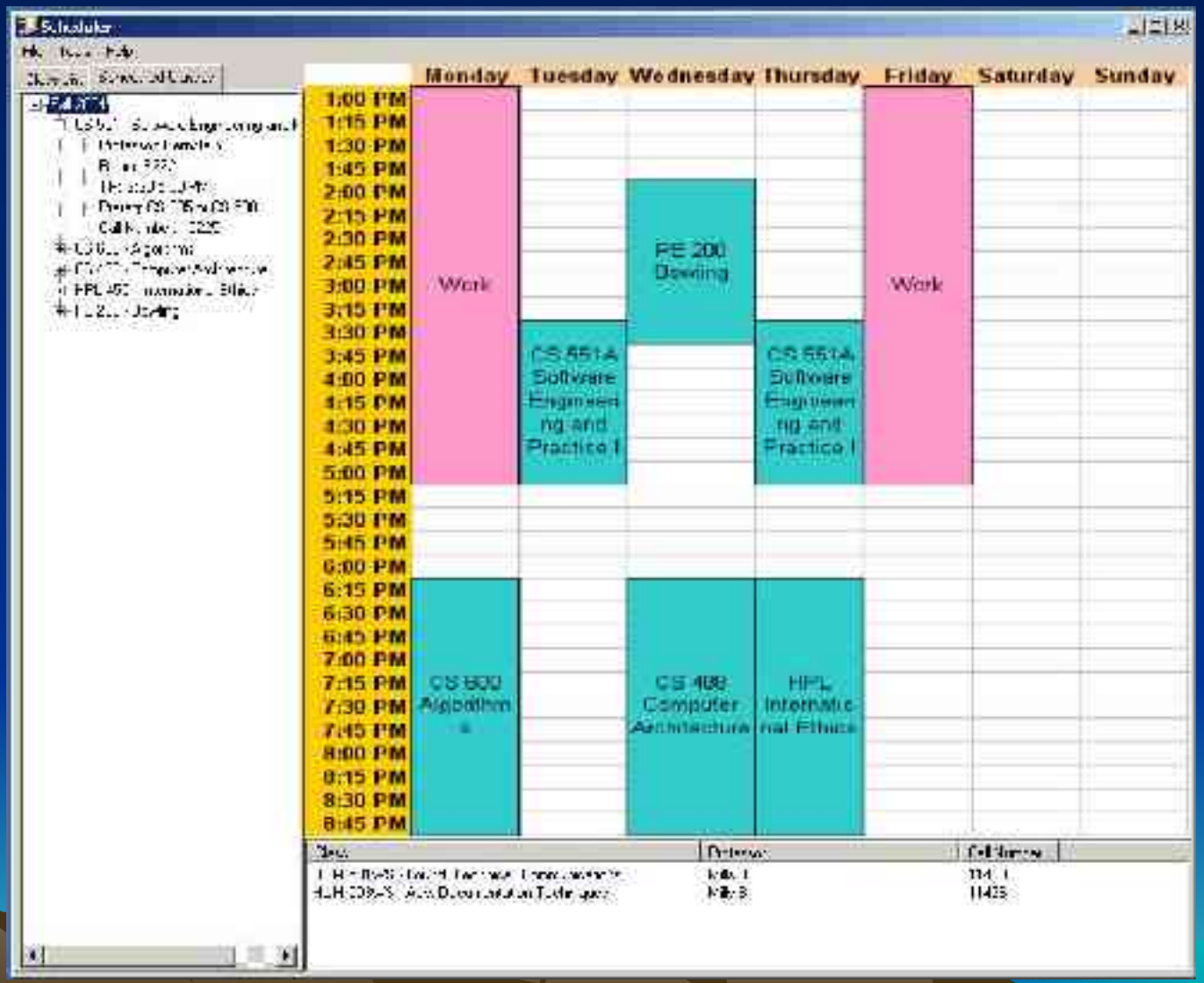
Requirements Case: SchedulerPro Prospectus

User friendly, efficient interface for students to create and modify class schedules.

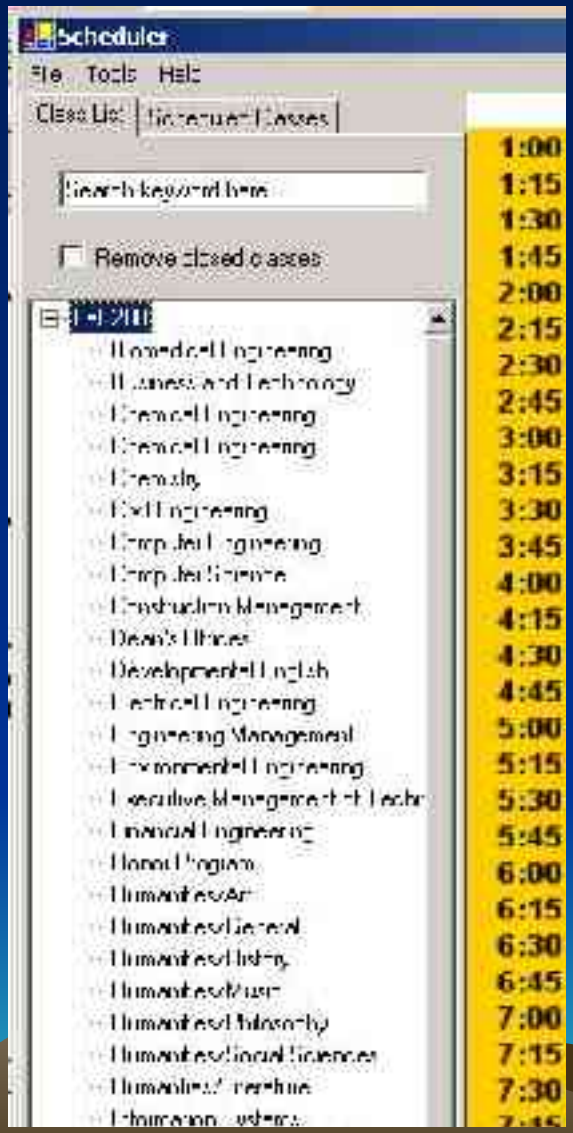
Features:

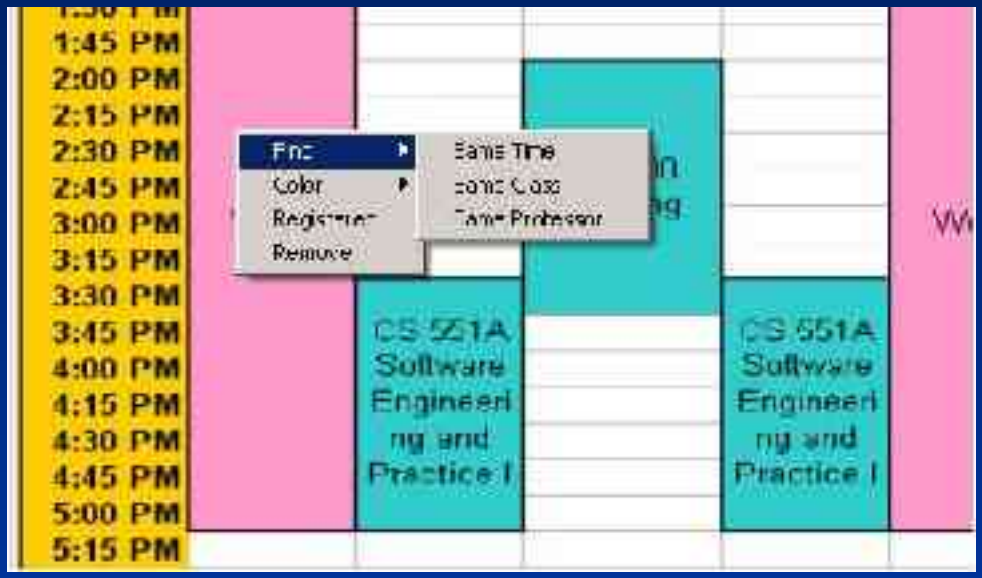
- Visual schedule creation and editing
- Schedule suggestion
- Schedule comparison view
- Monitor closed-out sections

SchedulerPro Prototype Screen



SchedulerPro Prototype Screen





SchedulerPro Notification Emails

From: schedulerpro@stevens.edu
Sent: Tuesday, April 19, 2005 12:15 PM
To: gdenngel@stevens.edu
Subject: Notification From Scheduler Pro

Scheduler PRO

This is an automated notification from Scheduler Pro. The following class is available for registration:

Notification Class Details

Title: Microprocessor Sys. Lab

Section: CS391C
Call Number: 10239
Instructor: STAFF
Scheduled Meetings: Thursday
Time: 11:00 AM-1:40 PM

Section: CS391D
Call Number: 10240
Instructor: STAFF
Scheduled Meetings: Thursday
Time: 2:00 PM-4:50 PM

Section: CS391A
Call Number: 10237
Instructor: STAFF
Scheduled Meetings: Tuesday
Time: 2:00 PM-4:50 PM

Section: CS391B
Call Number: 10238
Instructor: STAFF
Scheduled Meetings: Wednesday
Time: 8:00 AM-12:50 PM

This address is not monitored so please do not respond to this message. To discontinue this notification or to manage your schedule please visit the Scheduler Pro Homepage.

Measurable Operational Value SchedulerPro MOV

Reduce student withdrawals by 20%

SchedulerPro Functional Goals

Schedule Classes and Personal Time

- → Searching
- → Course Placement
- → Course Detail Viewing
- → Course Removal
- → Scheduling Personal Blocks
- → Notification (optional)
- → Course Suggestions (optional)

Functional Requirements

- Search available classes by:
  - ✓Same professor
  - ✓Similar time
  - ✓Same or equivalent class but different sections
- Register and track registrations
- Color classes and arbitrary time-blocks by user choice

SchedulerPro Nonfunctional Requirements

- Integrate with “Web for Students’ and existing authentication systems and avoid incompatibilities
- Allow schedules to be saved/accessed from a server or local file
- Provide a scaled time-accurate visual representation of the schedule

More Non-functional requirements

- Make schedules available even if the application is down, provided an internet connection is available
- Perform some functions without a live connection to the ‘Web for Students’ registrar web site

sQFD

Functions/ Features	Class Filters	Allocate non-class time	Long term information availability	Authenticate	
Makes scheduling classes easier	8	3	6	2 19	
Simplifies time management	7	9	8	2 26	
Find schedules in one place	1	1	5	7	14
Total	**16**	**13**	**19**	**11**	**59**

SchedulerPro Product Reliability

- Two hours of unavailability allows for daily backups, service, and reboots of the system
- Connections to server are minimized, reducing overall activity on the server.
- All sessions are a single web transaction.

SchedulerPro
Estimate of Reliability

$R(t) = 1 - F(t)$ $F(t) = P(T \leq t)$

- During load testing, we discovered the test server can support 1500 user queries a minute.
- P(failures/query) = 55/1500 = 0.036
- Thus, F(t) = 3.6%, which means the software is **96.4% reliable**

SchedulerPro Reliability Estimate

$1/\lambda = MTTF = \varepsilon E/kC$

k = scaling constant = 1

C is complexity = 2.78

E is the development effort = 36.4

ε is the expansion factor = 1.5

$\lambda = 0.05$

t is the continuous execution time for the software

$R(t) =$ **95.12%**

Complexity Chart - Client

- Project Type: online transaction

- Problem Domain: 2
- Architecture Complexity: 3
- Logic Design – Data: 2
- Logic Design – Code: 3
  - Total Score: 10
  - Complexity = (10/18) * 5 = 2.78

Complexity Chart - Server

- Project Type: online transaction

- Problem Domain: 1
- Architecture Complexity: 2
- Logic Design – Data: 2
- Logic Design – Code: 2
  - Total Score: 7
  - Complexity = (7/18) * 5 = 1.94

Complexity Chart - Overall

- Project Type: client/server
- Problem Domain: 2
- Architecture Complexity: 3
- Logic Design – Data: 2
- Logic Design – Code: 3
  - Total Score: 10
  - Complexity = (10/18) * 5 = 2.78

History of Function Points

Date	AFP	Project Length*	Projected Finish*
January 27	141	19.7 staff months	August
February 24	104	14.4 staff months	March
April 17	82	8.5 staff months	May

*Using COCOMO Model

Web Points

Function	Total Web Objects
Outputs	11
Inquiries	9
Inputs	18
Internal Files	31
External Interfaces	12
# Multimedia files	6
# Web building blocks	7
# Scripts	3
# Links	0
Total Web Objects	**97**

Effort = A * Π cd_i (Size)^{P1}

Because it is a small web project, assume:

A=2.1, B=2.0, P1=1.00, and P2=.5

Language Expansion Factor = 25

From this, estimate 2.43 KLOC

Effort = A* KLOC = 2.1 * 2.43

= 5.103 staff-months

Duration =B * Effort^{P2} = 2.0 * 5.103.5

= **4.518** calendar months

ICED-T

Scheduling by:	Intuitive	Consistent	Efficient	Durable	Thoughtful
Paper	3	2	2	2	3
Excel	3	2	3	3	3
School Scheduler	3	4	4	3	4
SchedulerPro	4	4	5	4	5

Creeping Featurism

- Endemic to the Software Industry
  - Occurs on more than 70% of all applications of over 1000 function points
- From a 60 project sample
  - Average creep was 35%
  - Maximum observed was 200%
  - Creeping requirements change about 1% per month
    - For a 3 year project, 1/3 of the delivered requirements would have been added after requirements were initially defined
- Rate of Requirements change is higher than for other forms of engineering (electrical, mechanical, civil)

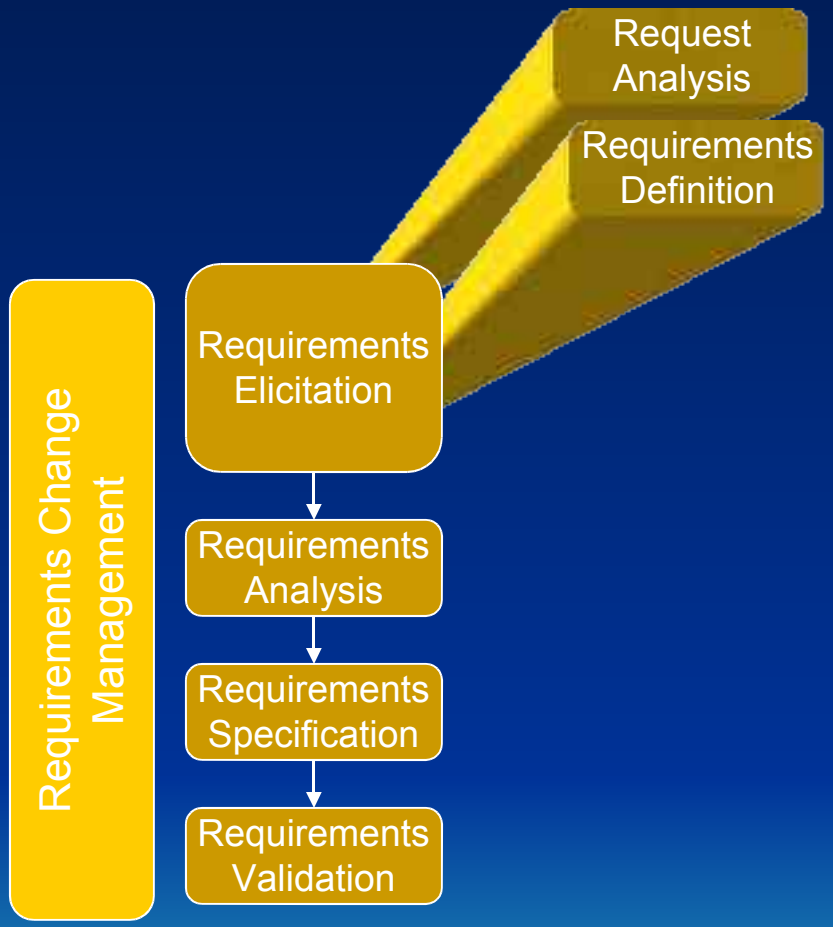
Root Causes of Creeping Requirements

- Uncertainty in resolving true user needs
- For multi-year projects, changes in normal business environment
- Failure to adopt methodologies that limit the risk associated with creeping requirements
- Primitive fundamental technologies for exploring and modeling requirements
- Failure to use technology to measure the impact of creeping requirements
- Engineering trade-off analysis is impossible

Requirements Management

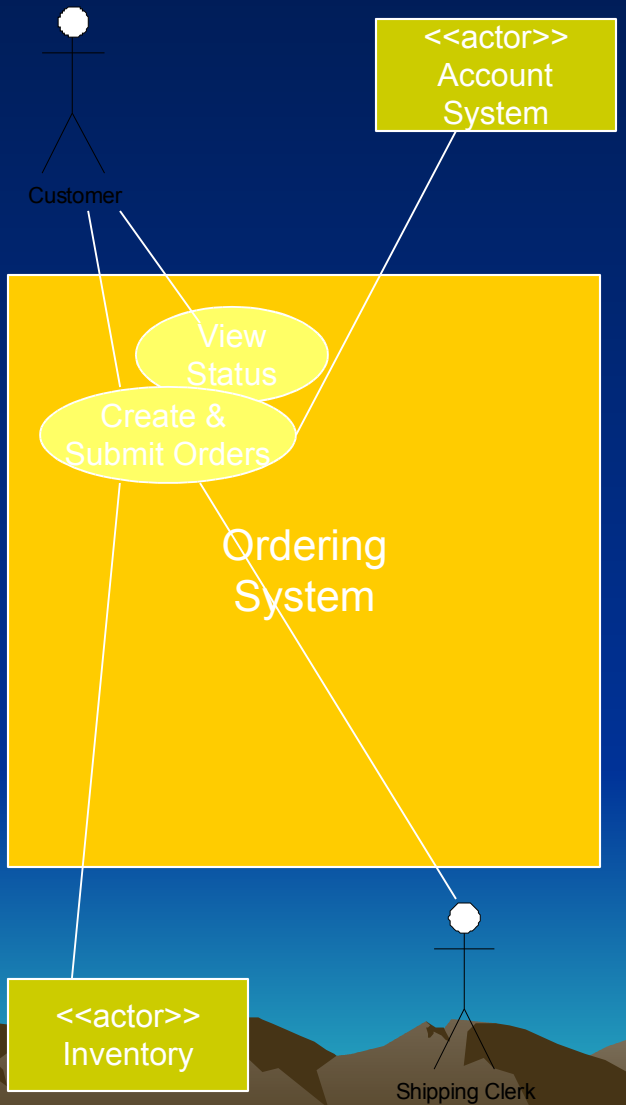
- Establish and maintain a business case to support funding
- Strategic linkages to business and technology organizations –AVOID SHELFWARE
- Continuous customer agreement on requirements
- Requirements agreement used as a basis for estimating, planning, implementing and tracking
- FORMAL COMMITMENT PROCESS

Requirements Process



- Elicitation
  - Request Analysis
    - Sourcing & Screening
  - Definition
    - Purposeful
    - Understand value
- Analysis
  - Interrelationships
  - Prioritization
  - Risk & Cost Assessment
- Specification
  - Modeling
- Validation
  - Agreement
- Change Management

Example

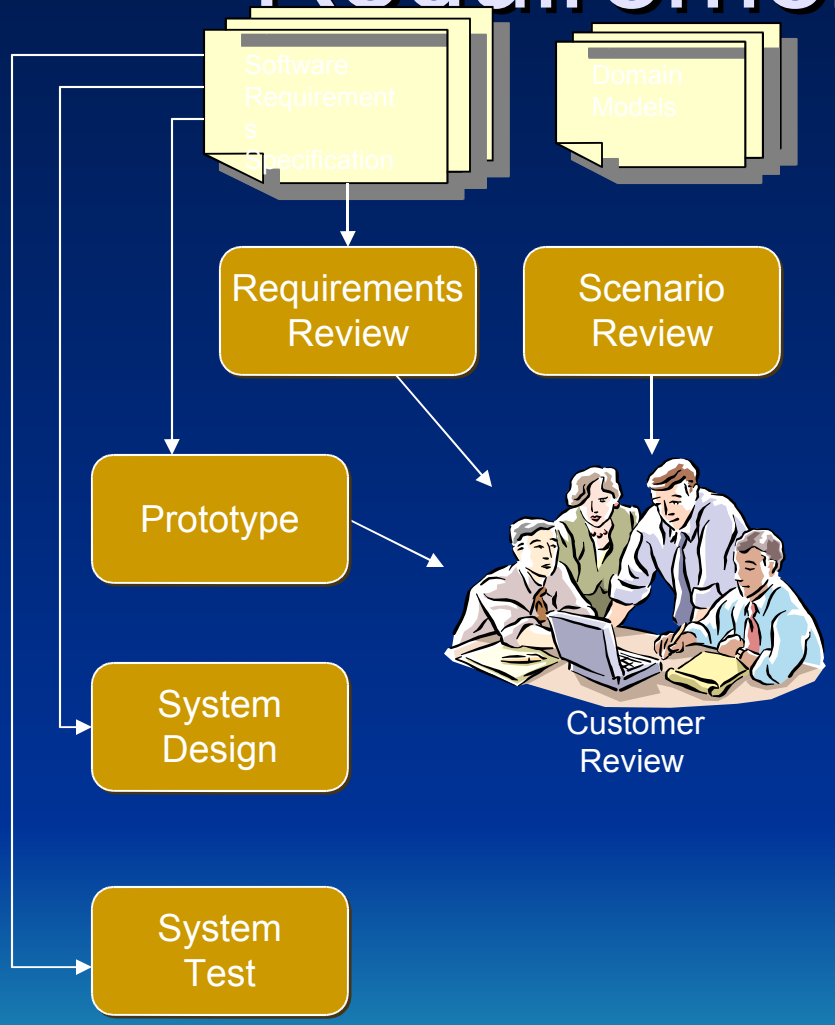


- Develop Use Cases
  - Focus on Goals
  - Identify Actors
  - Identify Main Tasks
- Use Case Concept
  - Complete, orthogonal, externally visible functionality
  - Initiated by an actor
  - Identifiable value to the actor

Requirements Specification Spec

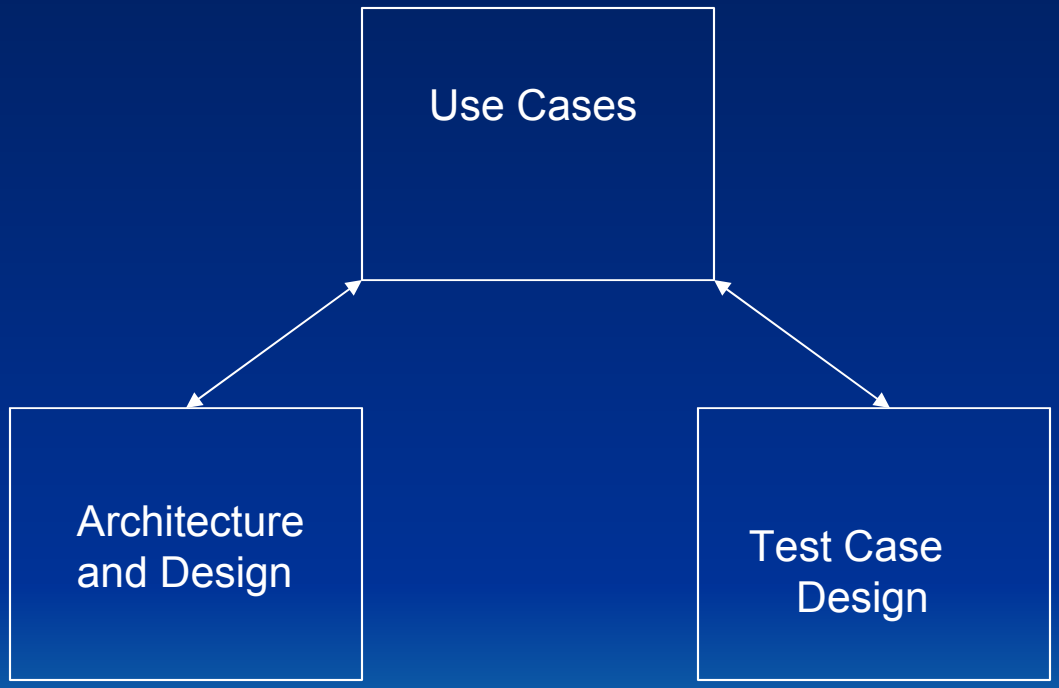
1. **Project Title, Revision Number and Author**
2. **Scope and Purpose of the system**
3. **Measurable Operational Value**
4. **Description**
5. **Feature List including ICED T and Simplified QFD analysis**
6. **Interfaces**
7. **Constraints**
8. **Change Log and Expected Changes**
9. **Responses to the unexpected**
10. **Measurements**
11. **Glossary**
12. **References**

Requirements Validation



- Requirements Reviews
  - Formal
  - Customer Representative
- **Prototyping**
- Model Validation
  - Scenario Reviews with Customers
  - Model Consistency
- Acceptance Tests
  - Verifiable Requirements

Use Cases Drive Development



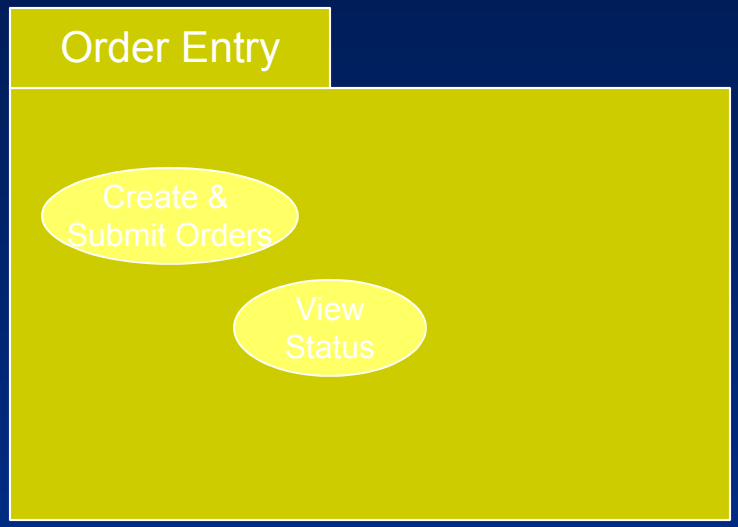
Use Case Documentation

Feature	Use Case
The customer can order on the web.	UC 1
The customer builds the order by selecting items from the on-line catalog and specifying a quantity.	UC 1
Only customers that have an account can create an order.	UC 1
At any time during the process of creating an order, the customer can determine the current price of the order.	UC 1
The customer signifies that the order is complete by submitting the order. When an order is submitted, it is assigned an order number.	UC 1
Customers with the priority privilege may designate an order as priority.	UC 1a
The customer can view the status of an order at any time by logging on to web site and requesting status on all open orders.	UC 2
Once an order is submitted, it is checked to see if it is pre-paid or whether the customer has an account in good standing. If these conditions are not met, the order is held until the conditions are met or the order is cancelled.	UC 1
…	

Use Case Documentation

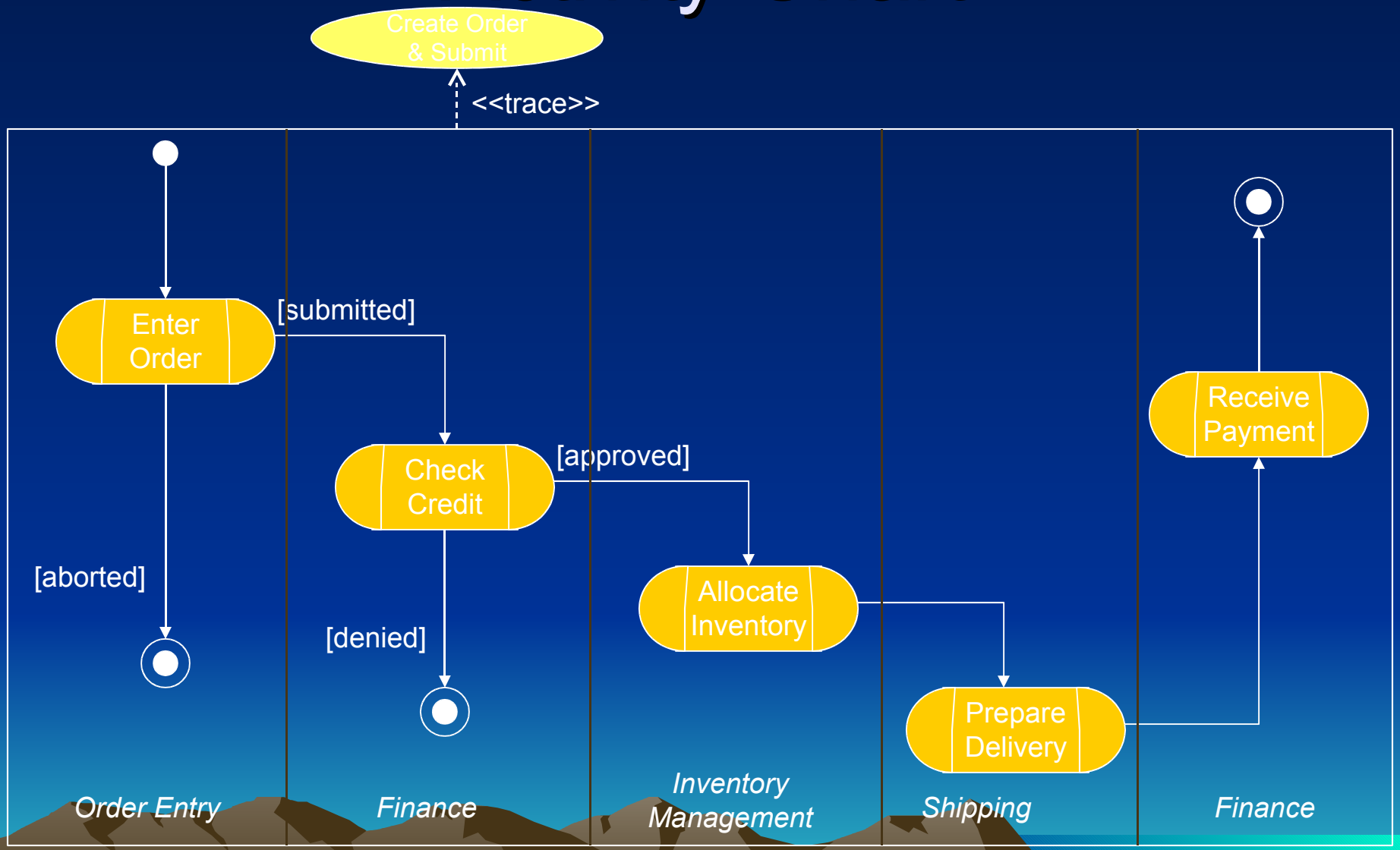
Use Case 1	Create Order & Submit
Brief Description	A customer wishes to order. Provided that the customer has a non-delinquent account or has pre-paid, the product is removed from inventory and delivered to the customer.
Actors	Customer, Inventory, Shipping Clerk, Account System
Trigger	Customer visits web site & creates an order.
Preconditions	Customer has established and account. Customer email address is known. Customers are pre-designated to enter priority orders.
Main flow	Customer visits web site, signs on and is validated. Customer selects items from the online catalog and builds an order. Customer is appraised of current cost of order. Customer may denote that the order is a priority Customer submits order when done. A customer order number is assigned and the customer's credit and account status are checked. If credit is OK or the account shows pre-payment, then the order is sent to the inventory system. …..
Alternative flows	Priority Order Account is delinquent. Action taken ? Cancelled ? Changes to or cancellation of the order? Order cannot be fulfilled ?
Postconditions	Order has been created and is either been cancelled or been fulfilled.

Package Diagram

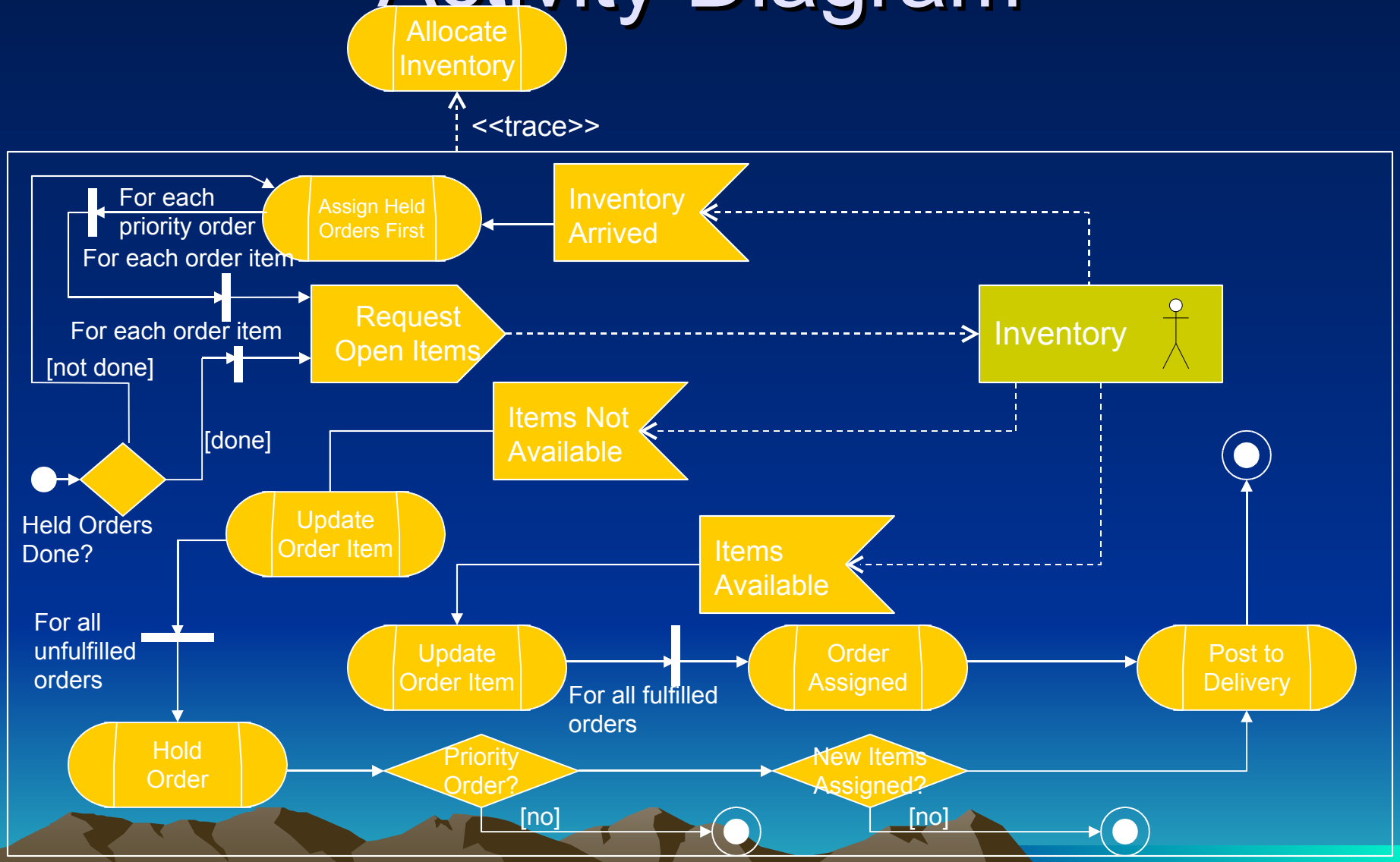


- Groups related use cases
- Forms basis for a functional partitioning from the users point of view.
- Shorthand for tracking within the project

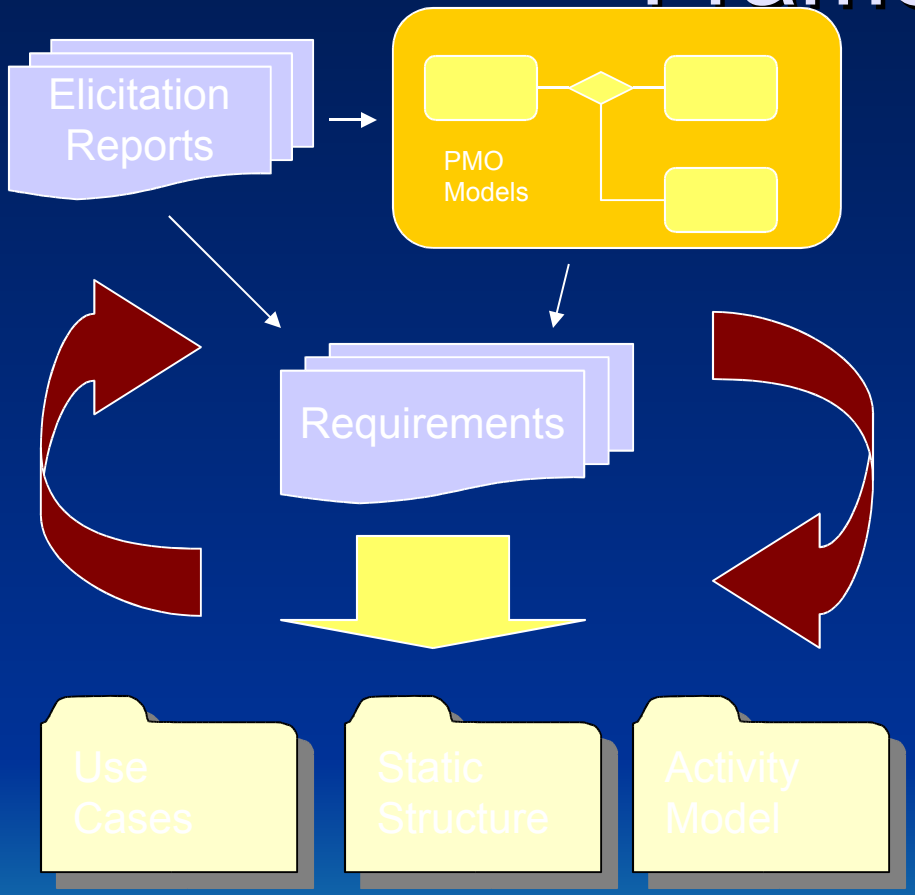
Activity Chart



Activity Diagram



Mapping Requirements to a Framework



- ICED T
  - Intuition
  - Consistent
  - Efficient
  - Durable
  - Thoughtful
- UML Framework
  - Use Cases
  - Structure
  - Business Rules

People

Software Trustworthiness depends on people:

I propose that customers insist that software products identify a Software Architect and Software Project Manager in their contracts

Software Architect:

- Affirms that the software product solves the customer's problem
- Affirms that the software product is suitably reliable, easy-to-use, extendible, not harmful and robust. That it is trustworthy.
- Affirms that the requirements are valid.

Software Project Manager:

- Affirms that the software was successfully tested against the requirements.
- Affirms and identifies the good software engineering processes were used in the software development and integration.
- Affirms that the project is within budget, on-time and performs satisfactorily.

Current Technologies Demanding Trustworthiness

- Interchangeable software components
- Applets
- Service Oriented Architecture
- Systems-of-Systems
- Web based distributed processing
- Peer-to-Peer computing

Systems Engineering

Systems Engineering

"An interdisciplinary approach and means to enable the realization of successful systems."

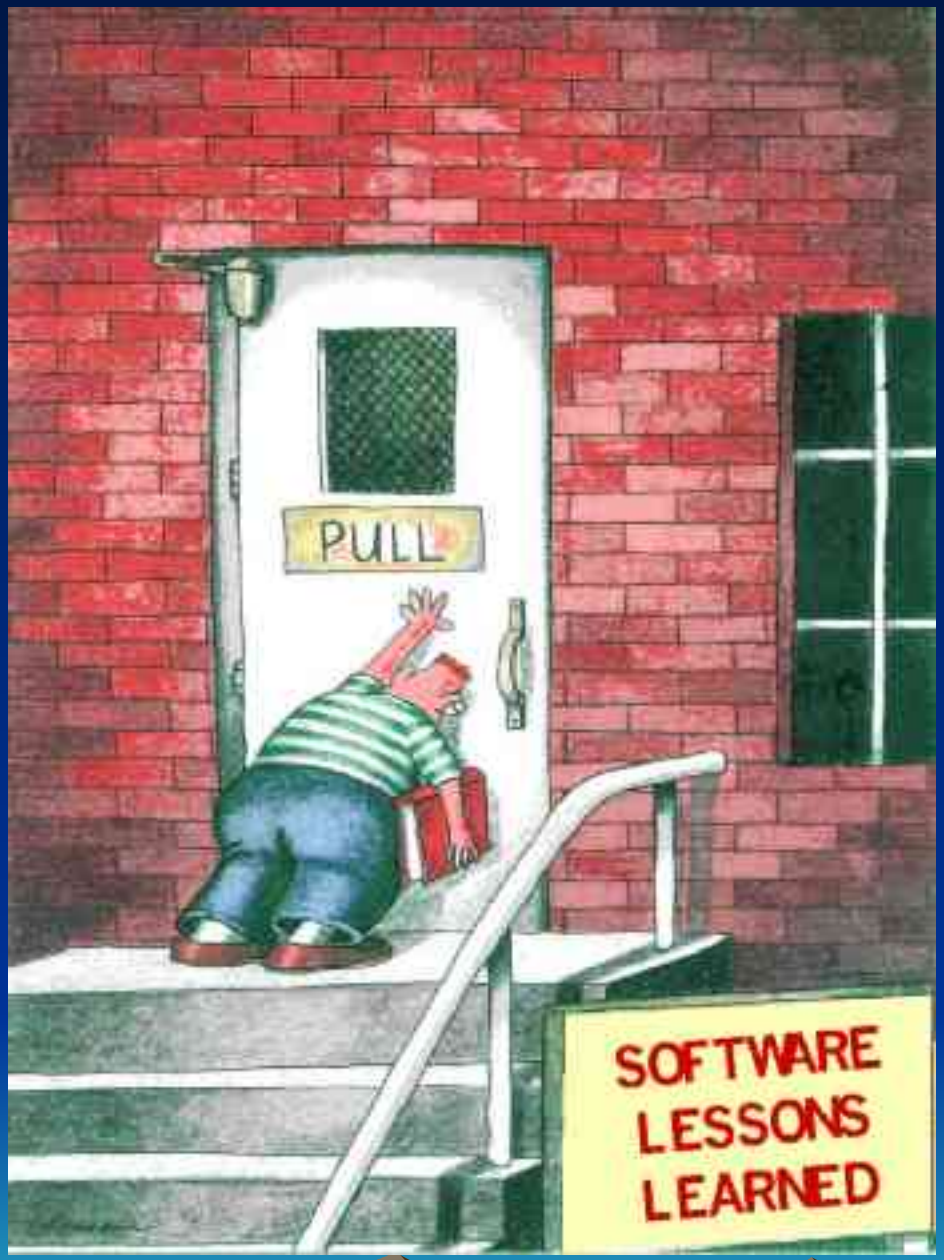
- INCOSE (The International Council on Systems Engineering)

System:

"A group of interacting, interrelated, or interdependent elements that together form a complex whole."

- NGE Project (Next Generation Education Project)

Systems Engineer


PULL
SOFTWARE
LESSONS
LEARNED