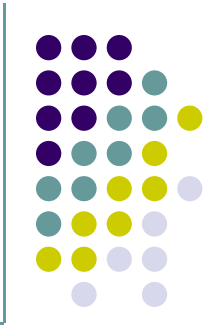
Professor Hausi A. Müller PhD PEng FCAE
Department of Computer Science
Faculty of Engineering
University of Victoria

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

Announcements



- New class room as of Wed
  - MAC 288 (original one)
- Midterm rescheduled due to lab clash
  - Fri, Feb 26 in class confirmed !!
- Website
  - Due today
  - Submission: send link to hausimuller@gmail.com with Subject: SENG 321 Website
- Assignments/Deliverables
  - S0, C0, S1, C1 specs posted
  - Group website spec posted
- Projects
  - Original RFP posted again
  - Check for project websites

ULS Systems Solve Wicked Problems

- ***Wicked problem***
  An ill-defined design and planning problem having incomplete, contradictory, and changing requirements.
- Solutions to wicked problems are often difficult to recognize because of complex interdependencies.

- This term was suggested by H. Rittel & M. Webber in “Dilemmas in a General Theory of Planning,” *Policy Sciences 4, Elsevier* (1973)

- Wicked problems are problems that are not amenable to *analytic, reductionist analysis.*

ULS Systems Solve Wicked Problems

- ***Wicked problem***
  An ill-defined design and planning problem having incomplete, contradictory, and changing requirements.
- Solutions to wicked problems are often difficult to recognize because of complex interdependencies.

- This term was suggested by H. Rittel & M. Webber in “Dilemmas in a General Theory of Planning,” *Policy Sciences 4, Elsevier* (1973)

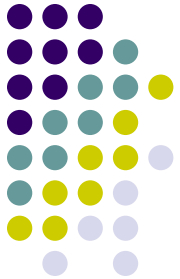
- Wicked problems are problems that are not amenable to *analytic, reductionist analysis.*

Characteristics of Wicked Problems

- You don't understand the problem until you have developed a solution
  - There is no definitive formulation of the problem.
  - The problem is ill-structured
  - An evolving set of interlocking issues and constraints
- There is no stopping rule
  - There is also no definitive Solution
  - The problem solving process ends when you run out of resources
- Every wicked problem is essentially unique and novel
  - There are so many factors and conditions, all embedded in a dynamic social context, that no two wicked problems are alike
  - No immediate or ultimate test of a solution
  - Solutions to them will always be custom designed and fitted

- Solutions are not right or wrong
  - Simply better, worse, good enough, or not good enough.
  - Solutions are not true-or-false, but good-or-bad.
- Every solution to a wicked problem is a one-shot operation.
  - You can't learn about the problem without trying solutions.
  - Every implemented solution has consequences.
  - Every solution you try is expensive and has lasting unintended consequences (e.g., spawn new wicked problems).
- Wicked problems have no given alternative solutions
  - May be no feasible solutions
  - May be a set of potential solutions that is devised, and another set that is never even thought of.

Good News



- Many software problems are not wicked

Bad News

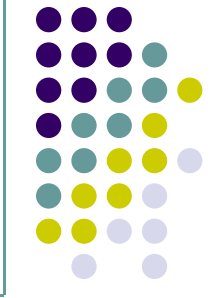
- Many software problems are wicked


Software
State of Practice

Common Problems ~20 Years Later

- Disappointed customers
- Serious quality issues
- Significant delays (years!)
- Canceled projects
- Deployed projects that are never used
- Rapidly changing requirements
- Long hours of overtime

Project Failures

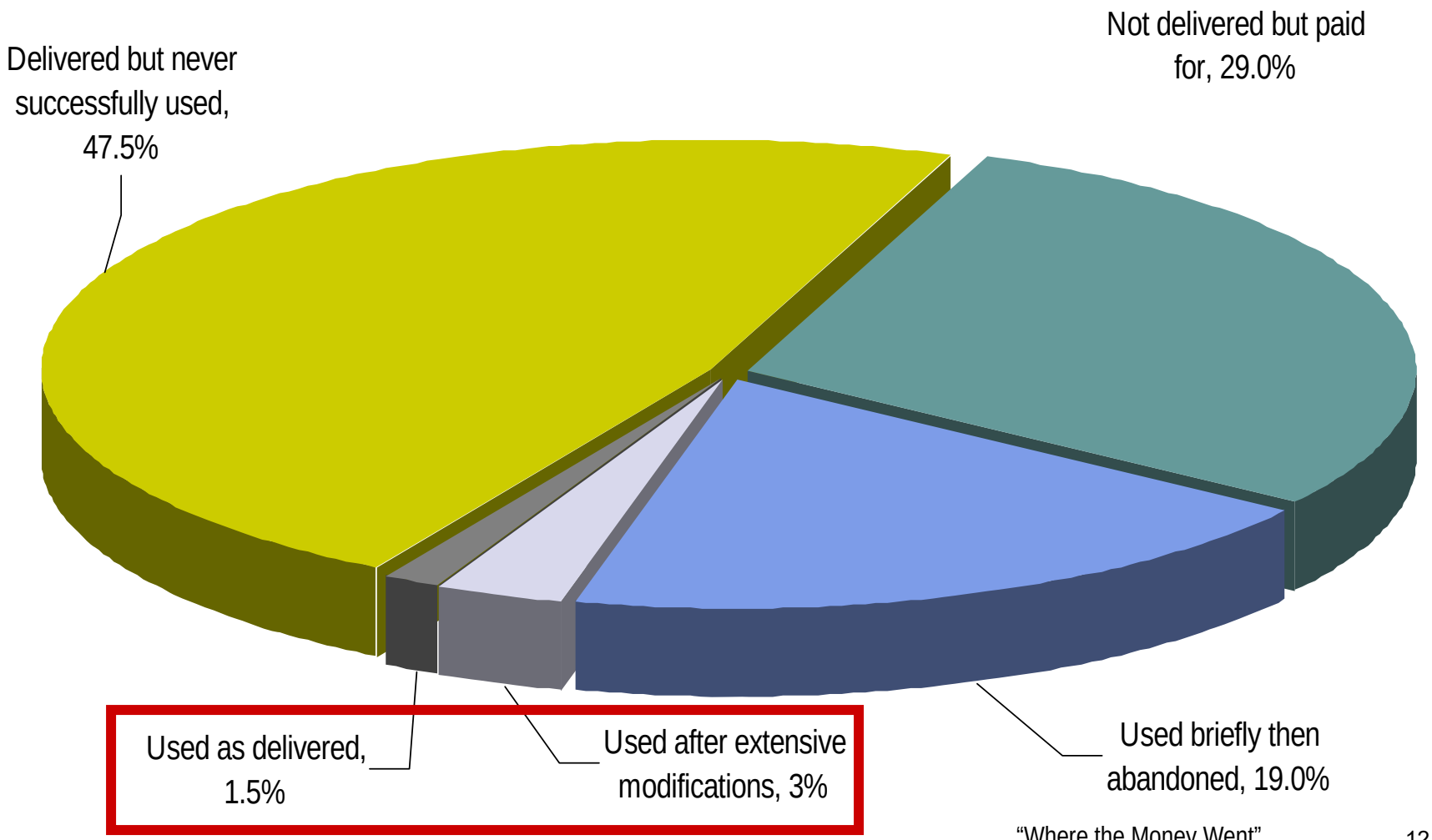


- Development teams spend insufficient time to understand *"the problem behind the problem"*
  - The real business problems—the big picture
  - The needs of the stakeholders (especially users)
  - The nature of the environment of the application
- The outcome is
  - Customer disappointment
  - Wasted resources
  - Systems that do not meet expectations

Asking the Right Questions

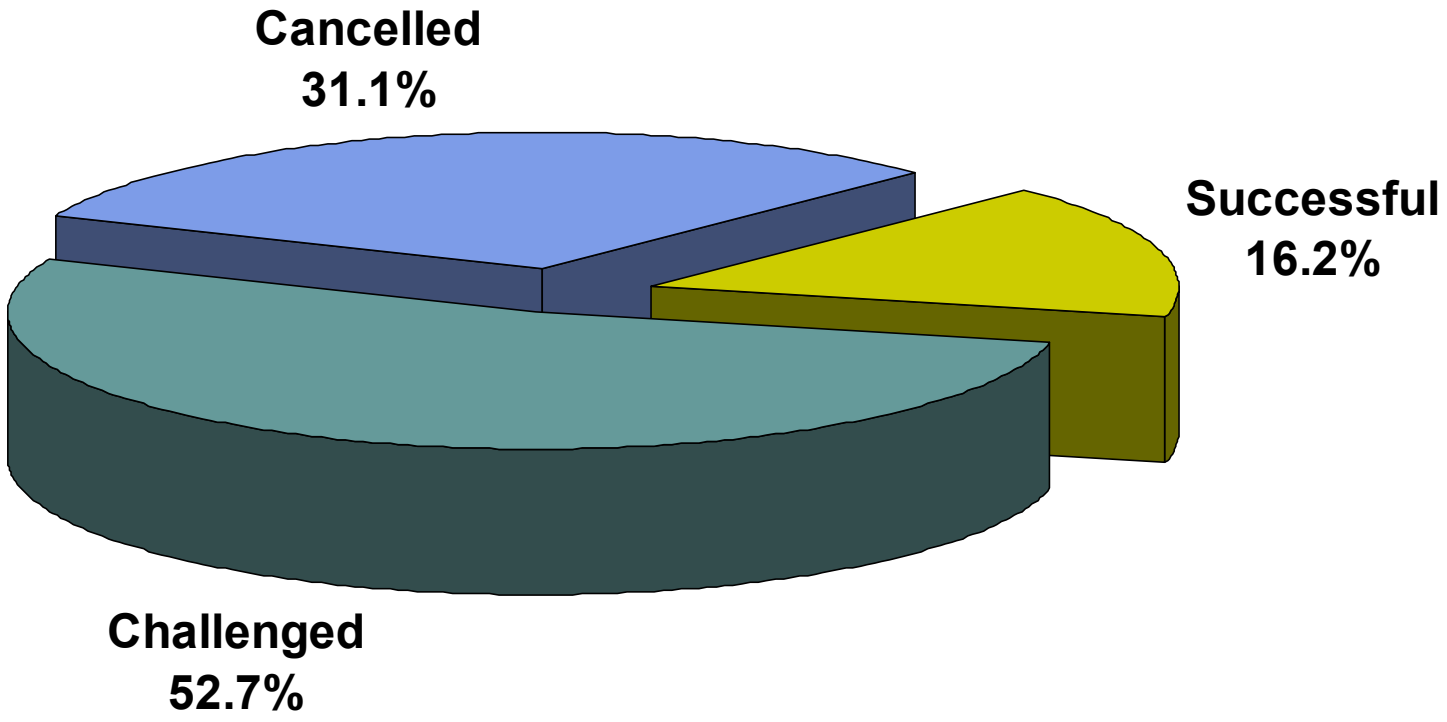
- The job of a requirement analyst is to
  - Ask the right questions
  - Understand the problem behind the problem
  - Determine the difference between What vs. How?
- Examples
  - Slow elevators in a skyscraper
  - Empty toothpaste boxes

Project Failures (1979)



"Where the Money Went"
Report to Congress by the Comptroller General, GAO, FGMSD80-4; Nov '79
(Based on 9 software contracts)

Project Failures (1995)

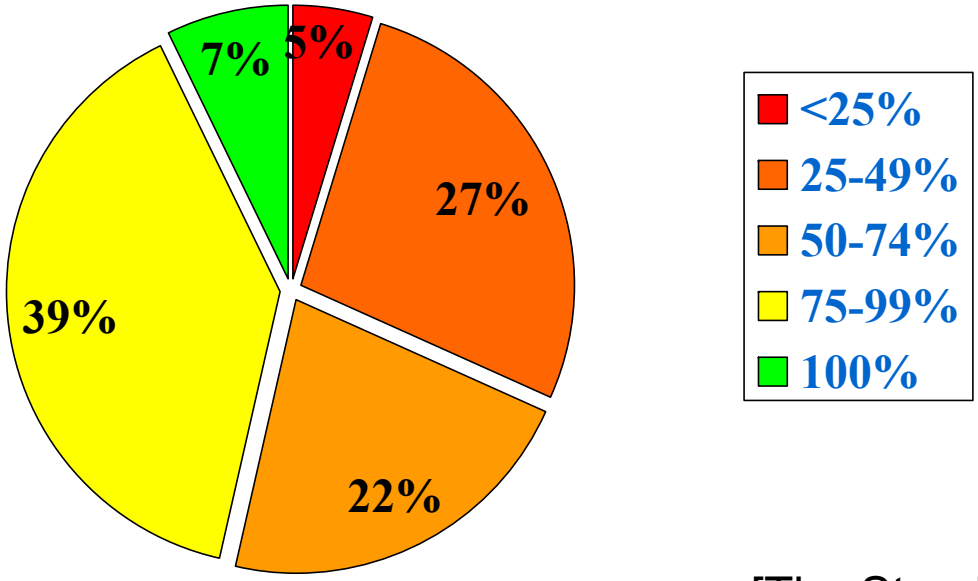


- Large companies had more canceled than challenged
- Small companies had more challenged than canceled

[The Standish Group, 1995]

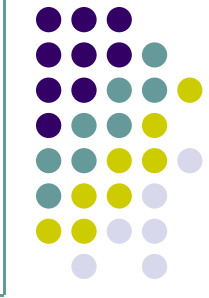
Delivered Functionality

- Only 7% of challenged projects deliver the originally planned functionality.
- Over a 50% chance to deliver just 50% of the originally planned functionality.



[The Standish Group, 1995]

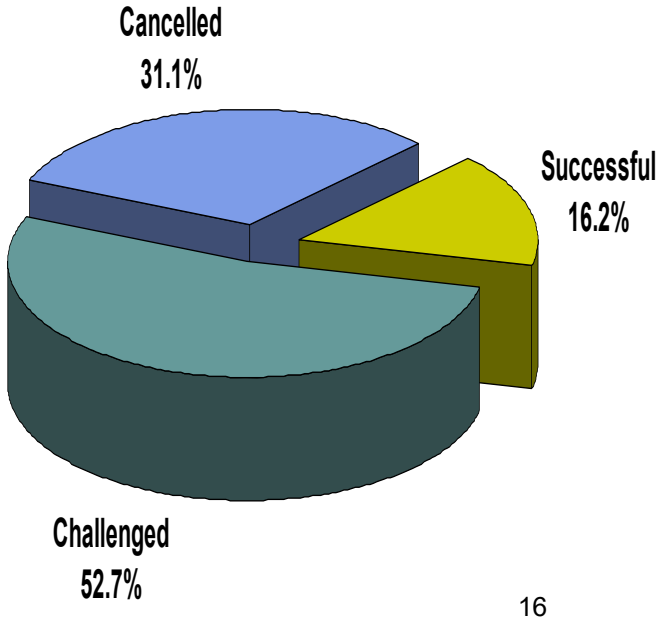
Cost of Incorrect or Incomplete Requirements



- [1981] ~75–85% of all errors found in SW can be traced back to the requirements and design phases
- [2000] Based on a survey of the cost of maintaining 500 major projects, 70–85% of total project costs are due to requirement errors and new requirements

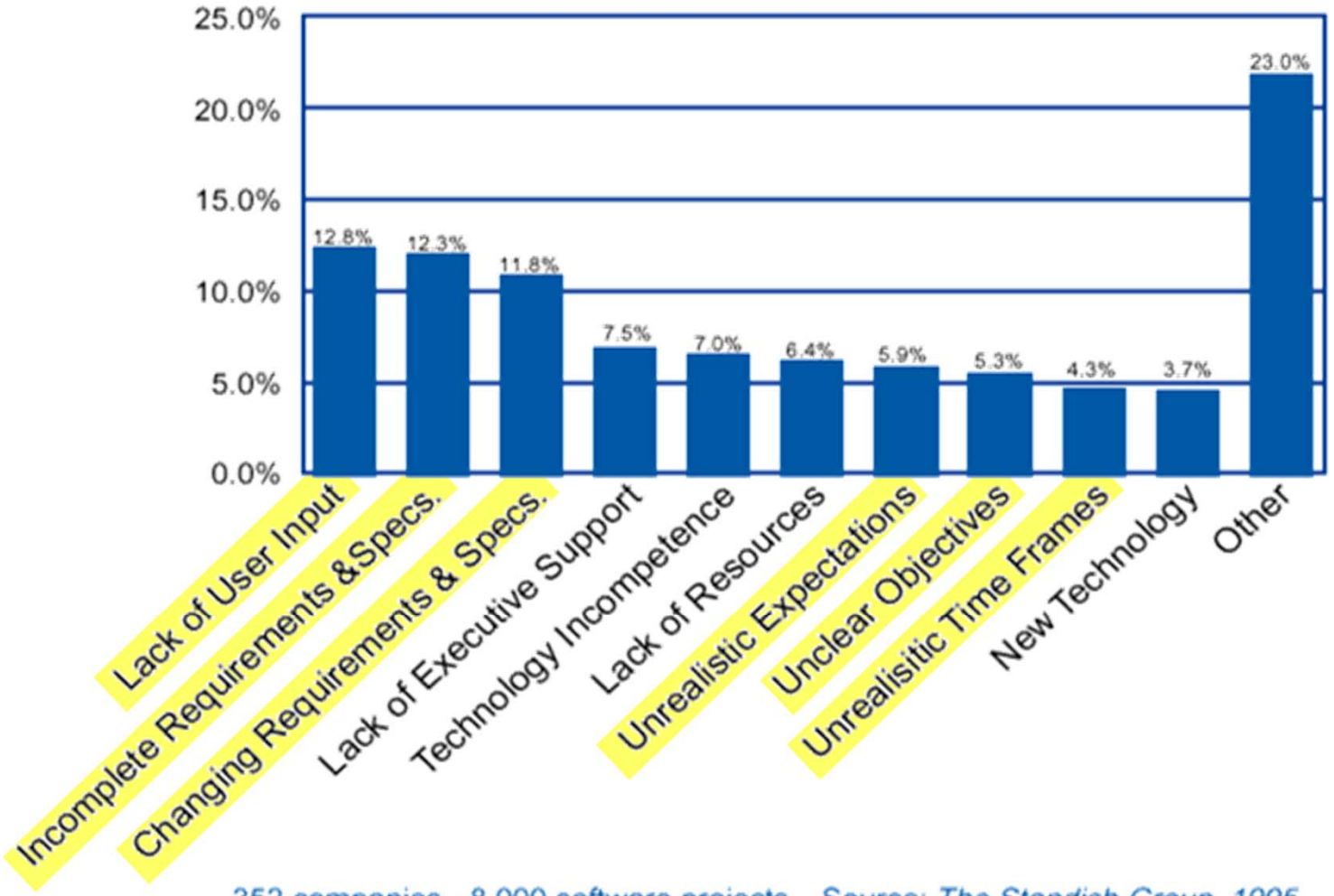
Root Causes for Project Failure and Primary Success Factors

- Root cause for Challenged projects:
  - Lack of user input: 13% of all projects
  - Incomplete requirements and specifications: 12%
  - Changing reqs and specs: 12%
- Primary success factors
  - User involvement: 16%
  - Executive support: 14%
  - Clear statement of reqs: 12%



[The Standish Group, 1995]

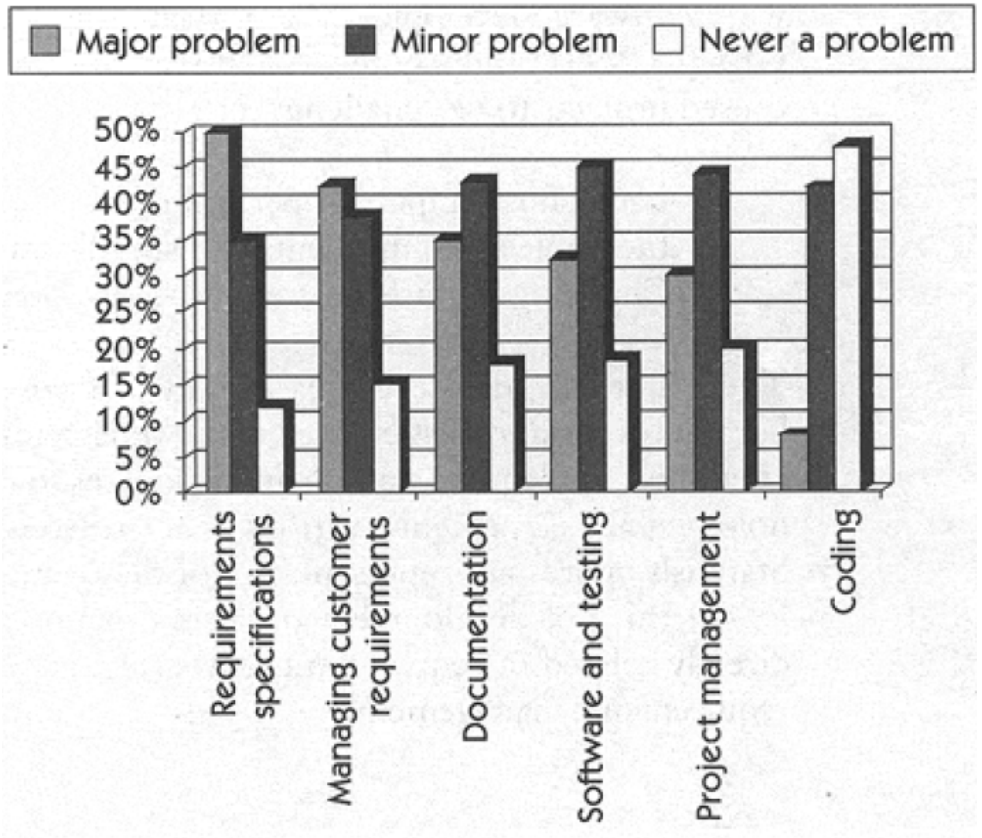
Why Software Projects Fail



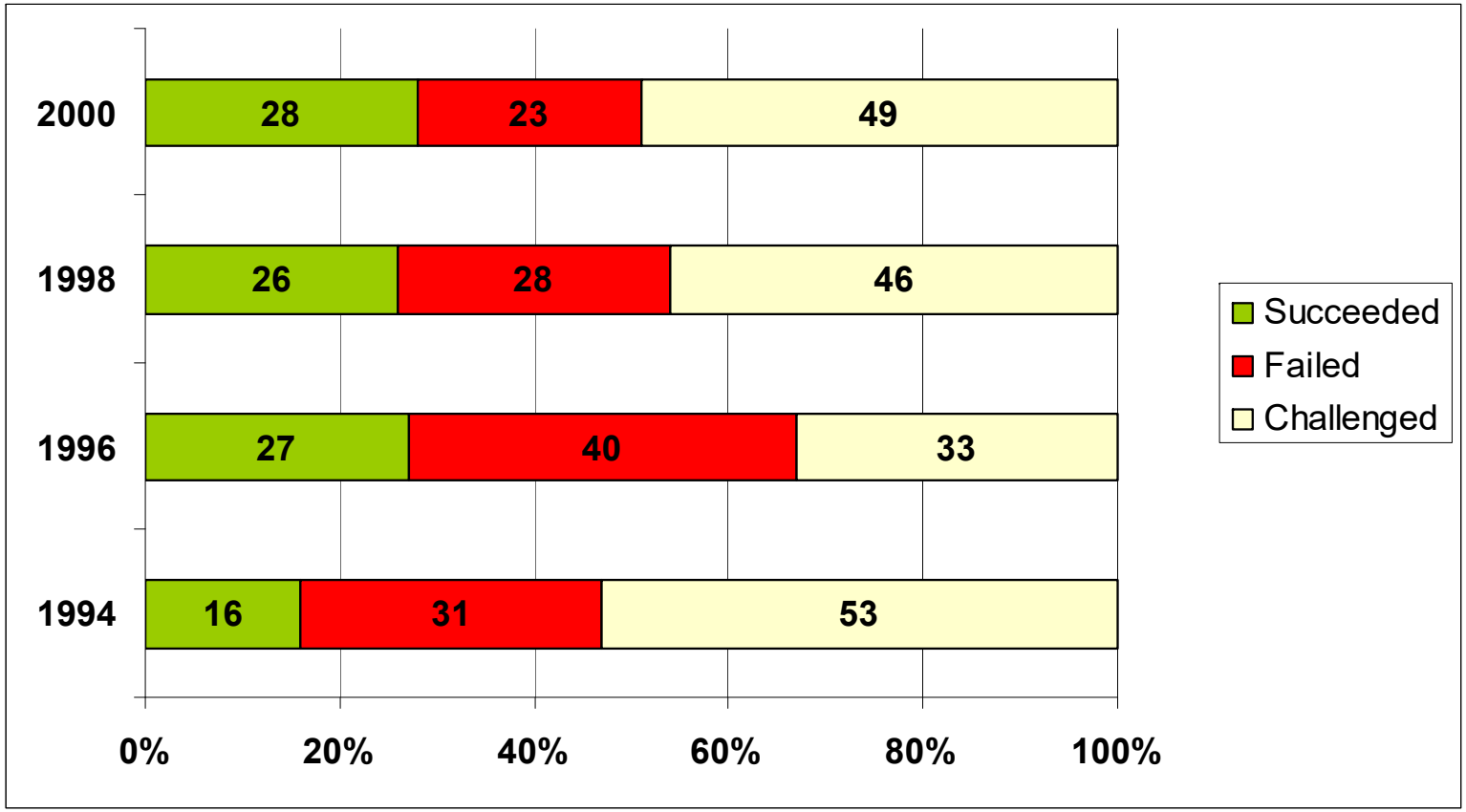
352 companies - 8,000 software projects. Source: *The Standish Group, 1995*

Largest Software Development Problems By Category

- The largest problems appearing in ~50% of responses:
  - Requirement specifications
  - Managing customer requirements
- Coding was a non-issue

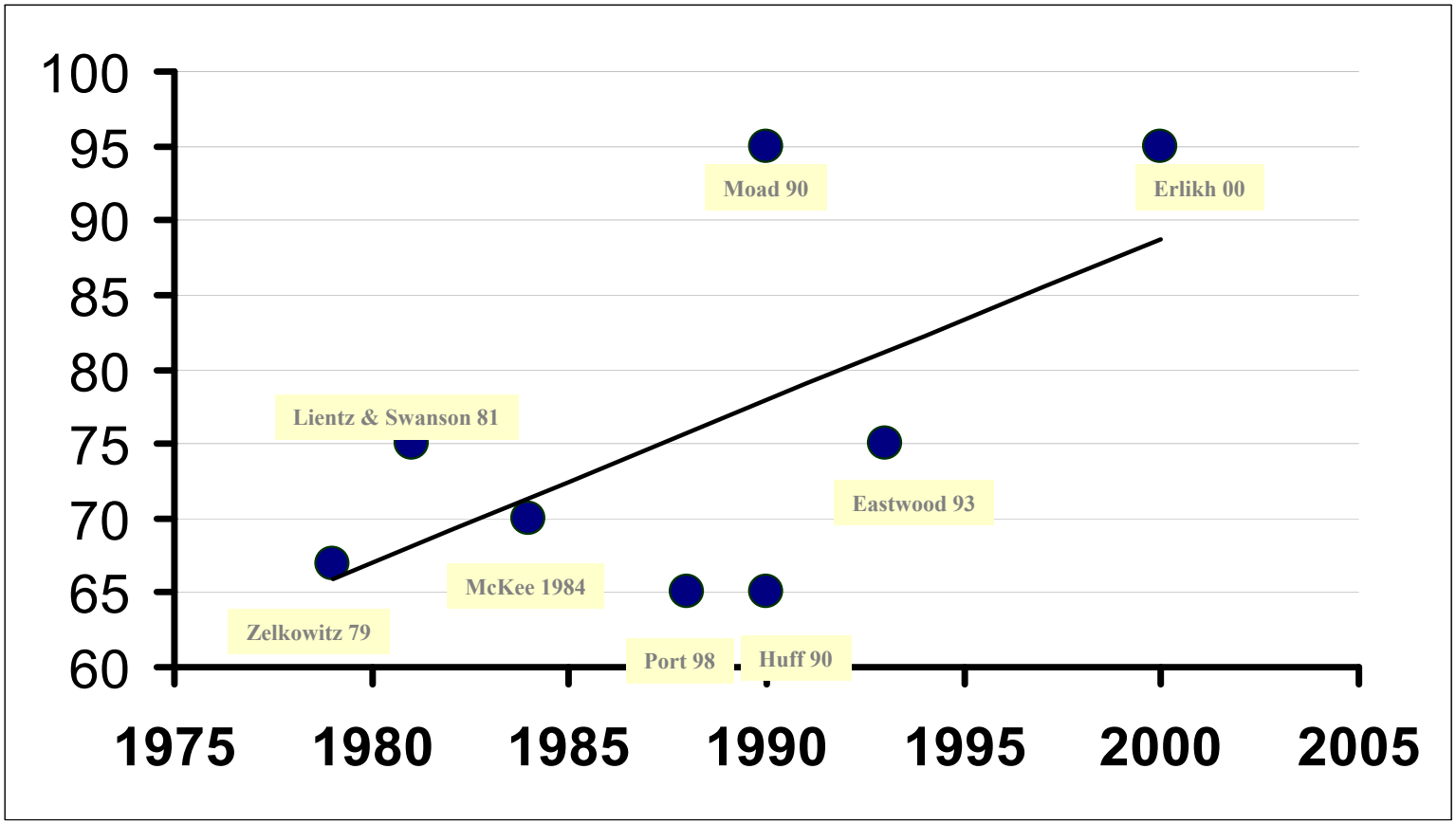


Projects Show Steady Slow Improvement



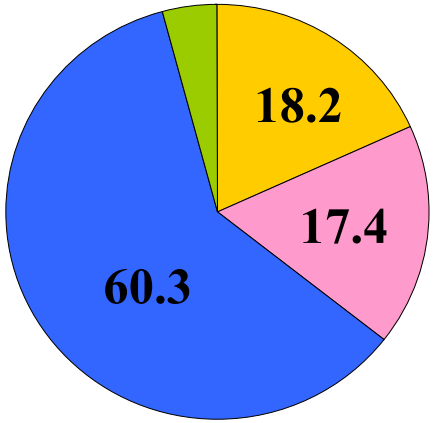
Based on projects in 30,000 US companies
[Standish Group 1994-2000]

Percentage of Project Costs Devoted to Maintenance

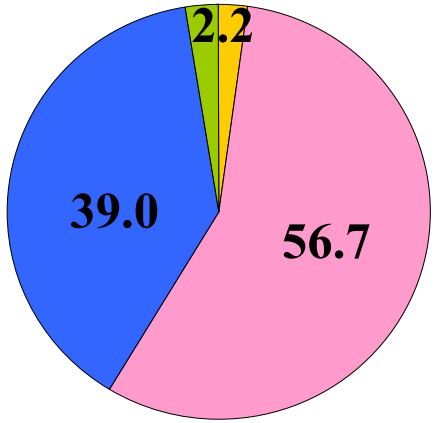


Survey of Software Maintenance Activities

- **Perfective:** add new functionality
- **Corrective:** fix faults
- **Adaptive:** new file formats, refactoring

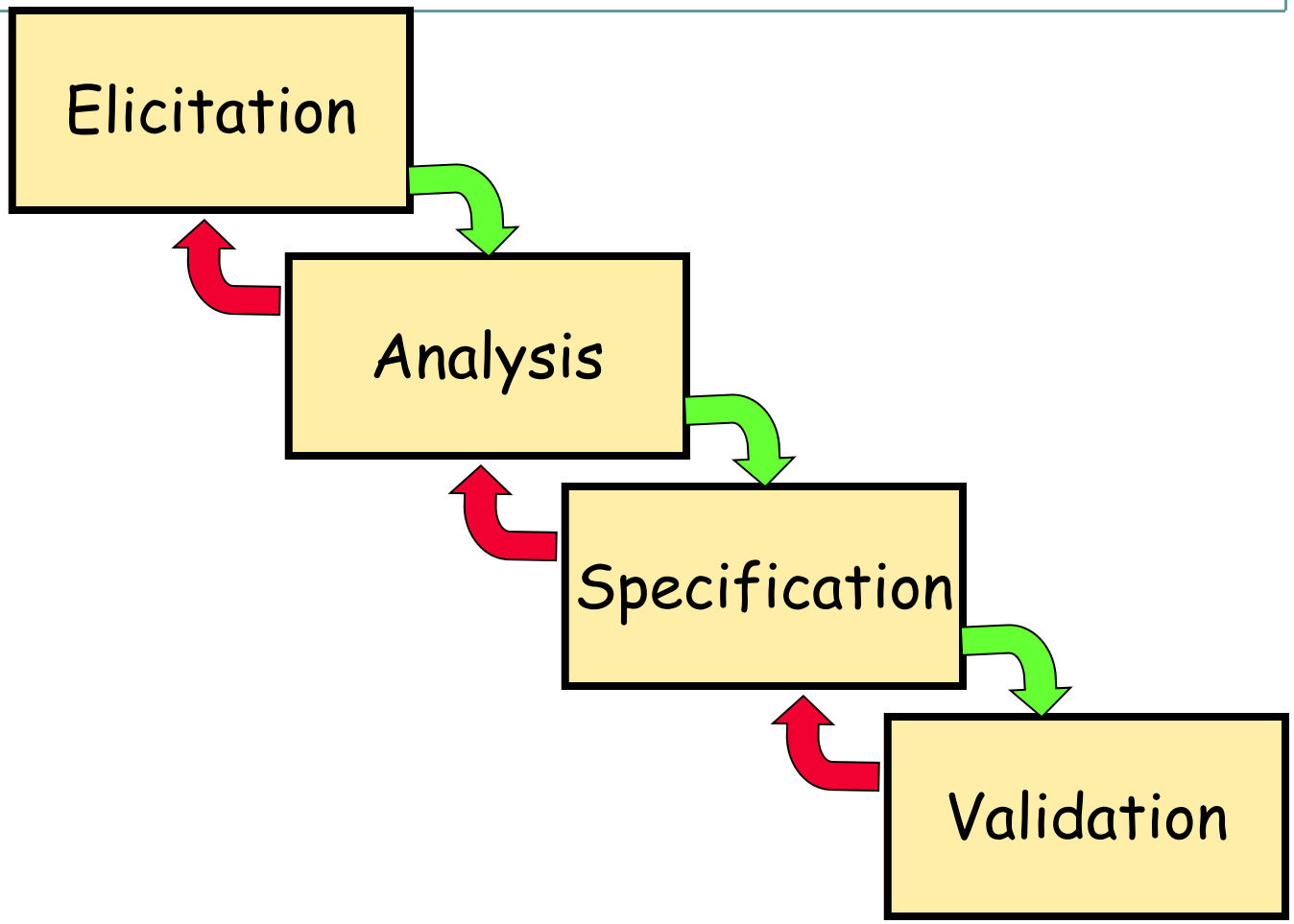


Lientz, Swanson, Tompkins [1978]
Nosek, Palvia [1990]
MIS Survey

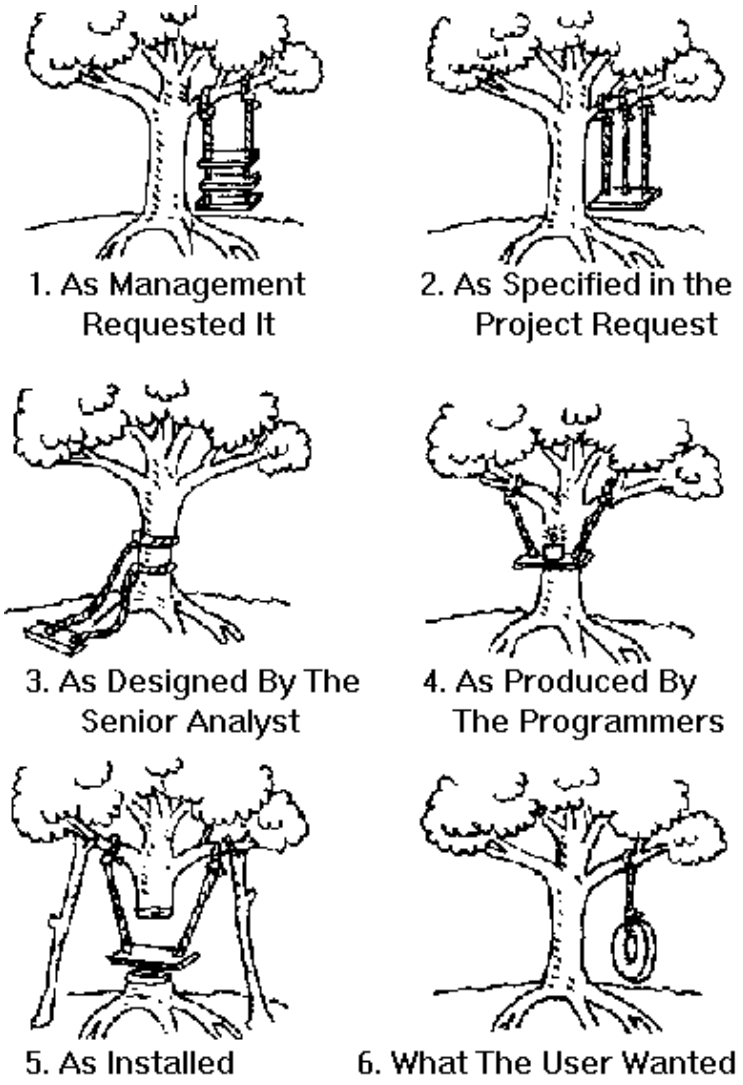


Schach, Jin, Yu, Heller, Offutt [2003]
Mining ChangeLogs
(Linux, GCC, RTP)

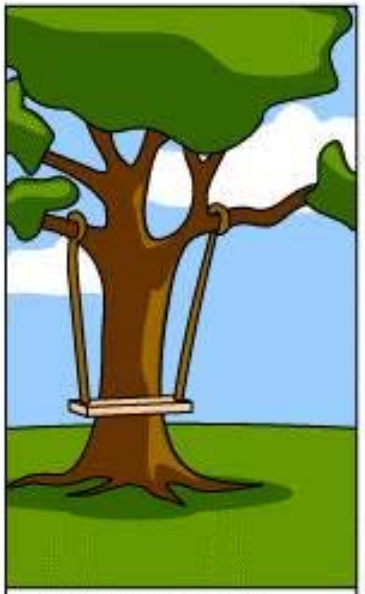
Requirement Engineering Process



Many Stakeholders: Different Visions and Conflicting Goals

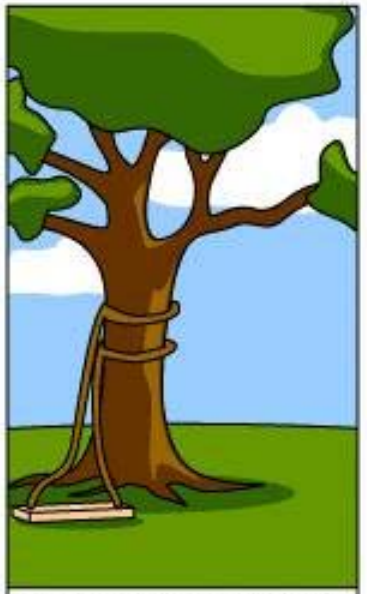


How the customer explained it



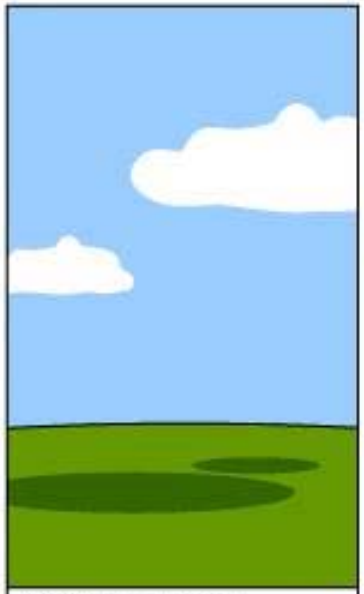
How the Project Leader understood it

How the Analyst designed it

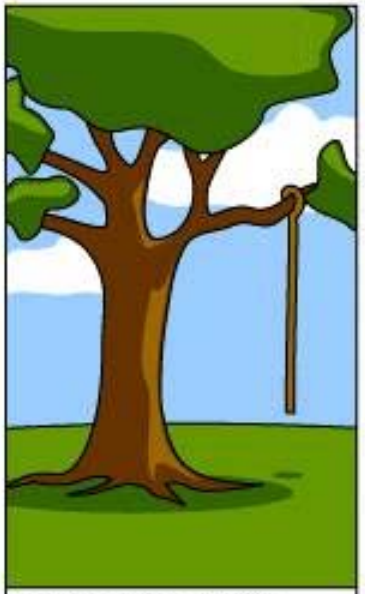


How the Programmer wrote it

How the Business Consultant described it

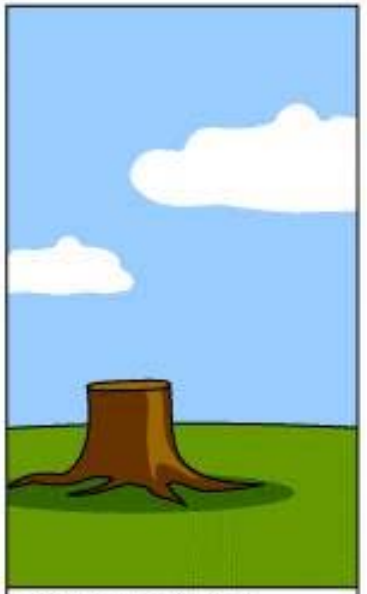


How the project was documented



What operations installed

How the customer was billed



How it was supported

What the customer really needed

Antagonism between Users and Developers

Developers' View of Users	Users' View of Developers
Users don't know what they want.	Developers don't understand operational needs.
Users can't articulate what they want.	Developers place too much emphasis on technicalities.
Users have too many needs that are politically motivated.	Developers try to tell us how to do our jobs.
Users want everything right now.	Developers can't translate clearly stated needs into a successful system.
Users can't prioritize needs.	Developers say no all the time.
Users refuse to take responsibility for the system.	Developers are always over budget.
Users are unable to provide a usable statement of needs.	Developers are always late.
Users are not committed to system development projects.	Developers ask users for time and effort, even to the detriment of the users' important primary duties.
Users are unwilling to compromise.	Developers set unrealistic standards for requirements definition.
Users can't remain on schedule.	Developers are unable to respond quickly to legitimately changing needs.