



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 !!**
- Assignments/Deliverables
 - S0, C0, S1, C1 specs posted
 - Group website spec posted
- Projects
 - Original RFP posted again
 - **Check for project websites**
- Website
 - **Due today**
 - Submission: send link to hausimuller@gmail.com with Subject: SENG 321 Website

2

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*.



3

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*.



4

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.



5

Good News

- Many software problems are not wicked

6

Bad News

- Many software problems are wicked

7



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

9

Project Failures

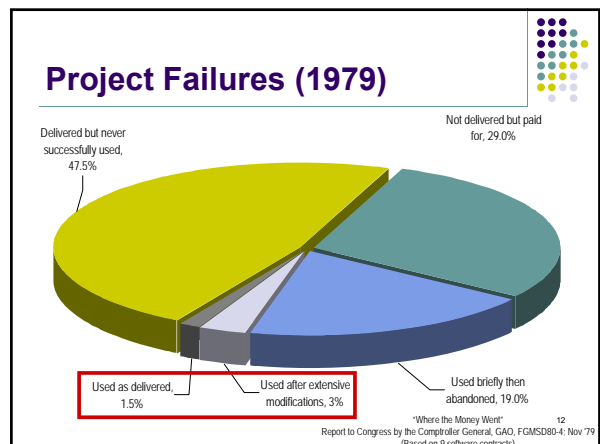
- Development teams spend insufficient time to understand *“the problem behind the problem”*
 - The real business problems—the big picture
 - The needs of the stakeholders (especially users)
 - The nature of the environment of the application
- The outcome is
 - Customer disappointment
 - Wasted resources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

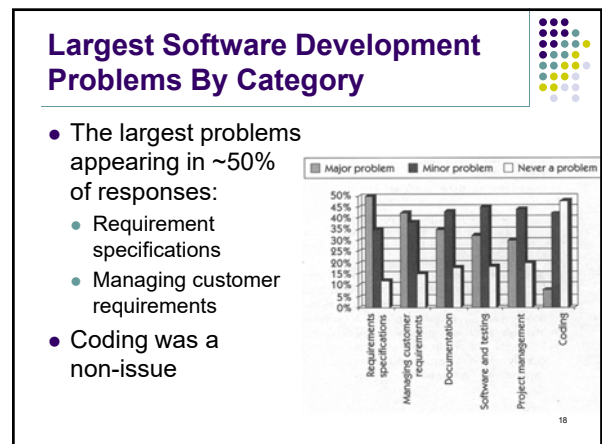
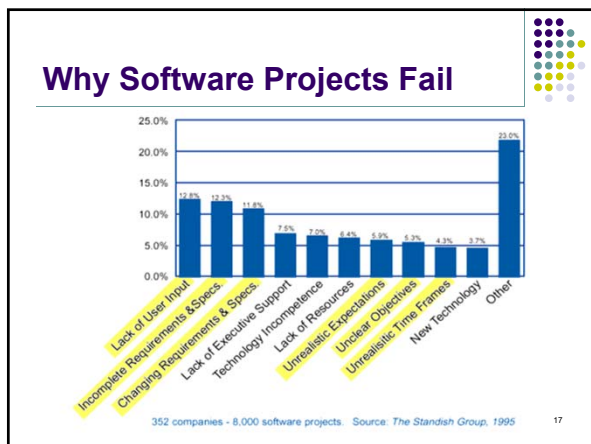
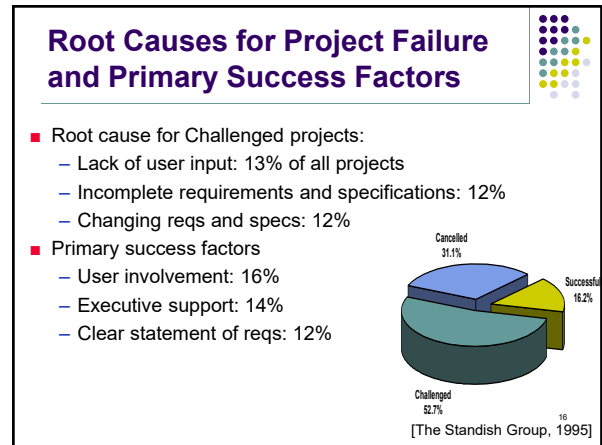
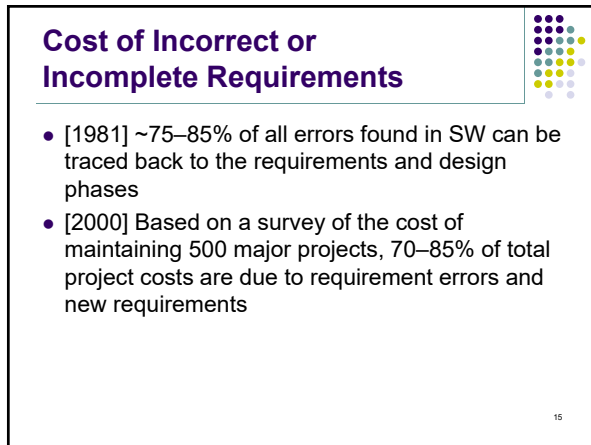
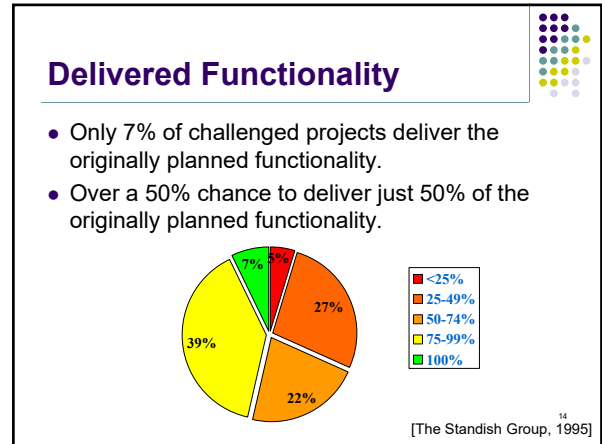
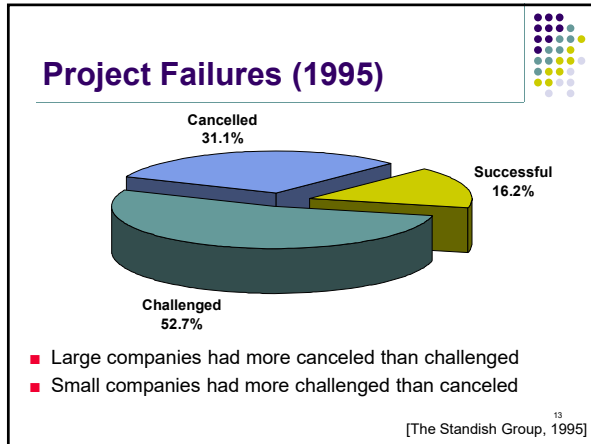
10

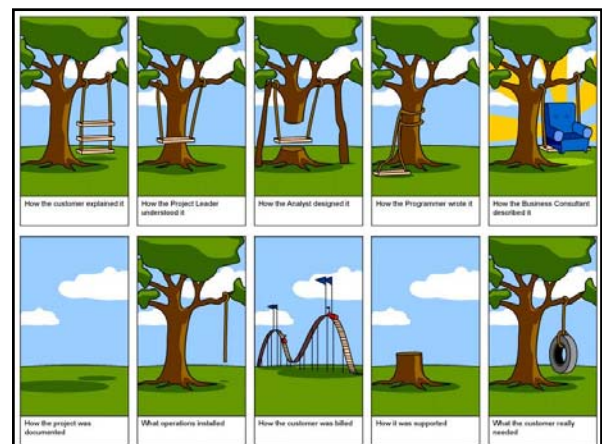
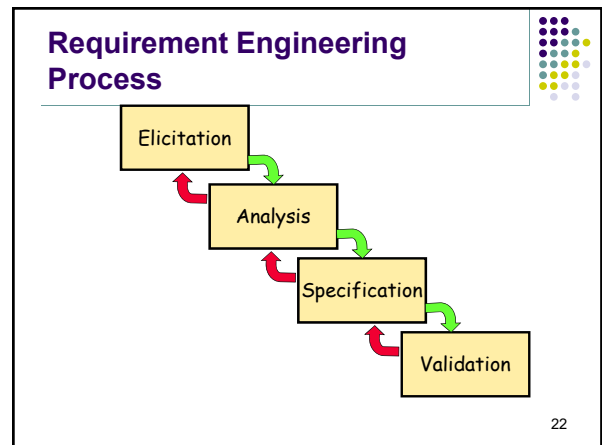
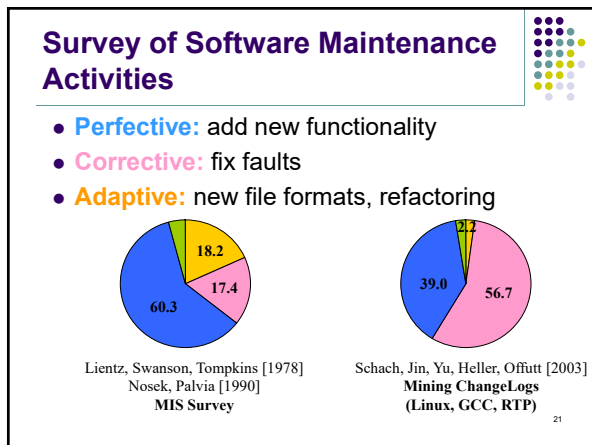
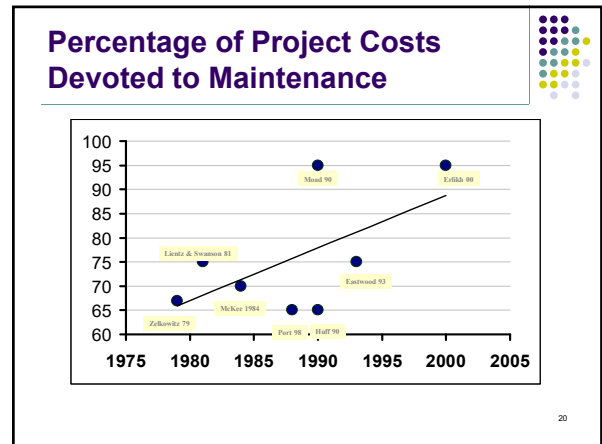
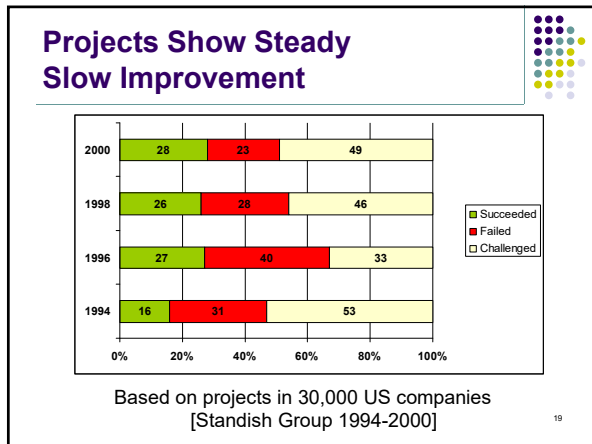
Asking the Right Questions

- 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

11







Antagonism between Users and Developers



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