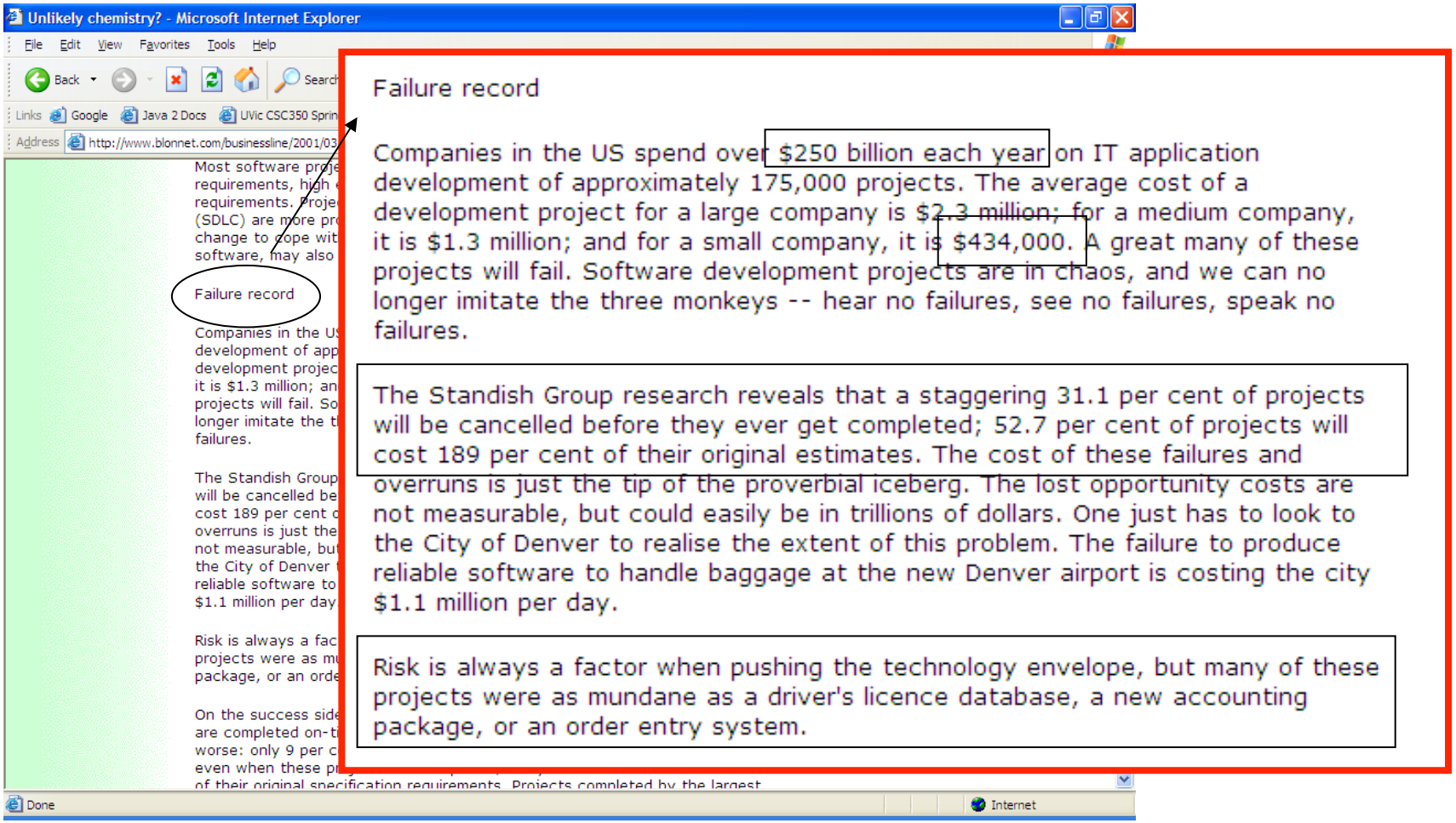
Software Engineering Process: Overview

- Some context, terms & concepts relating to software engineering
- The meaning of process
- Software lifecycle and its standard phases
- Several different software-process models

One reality

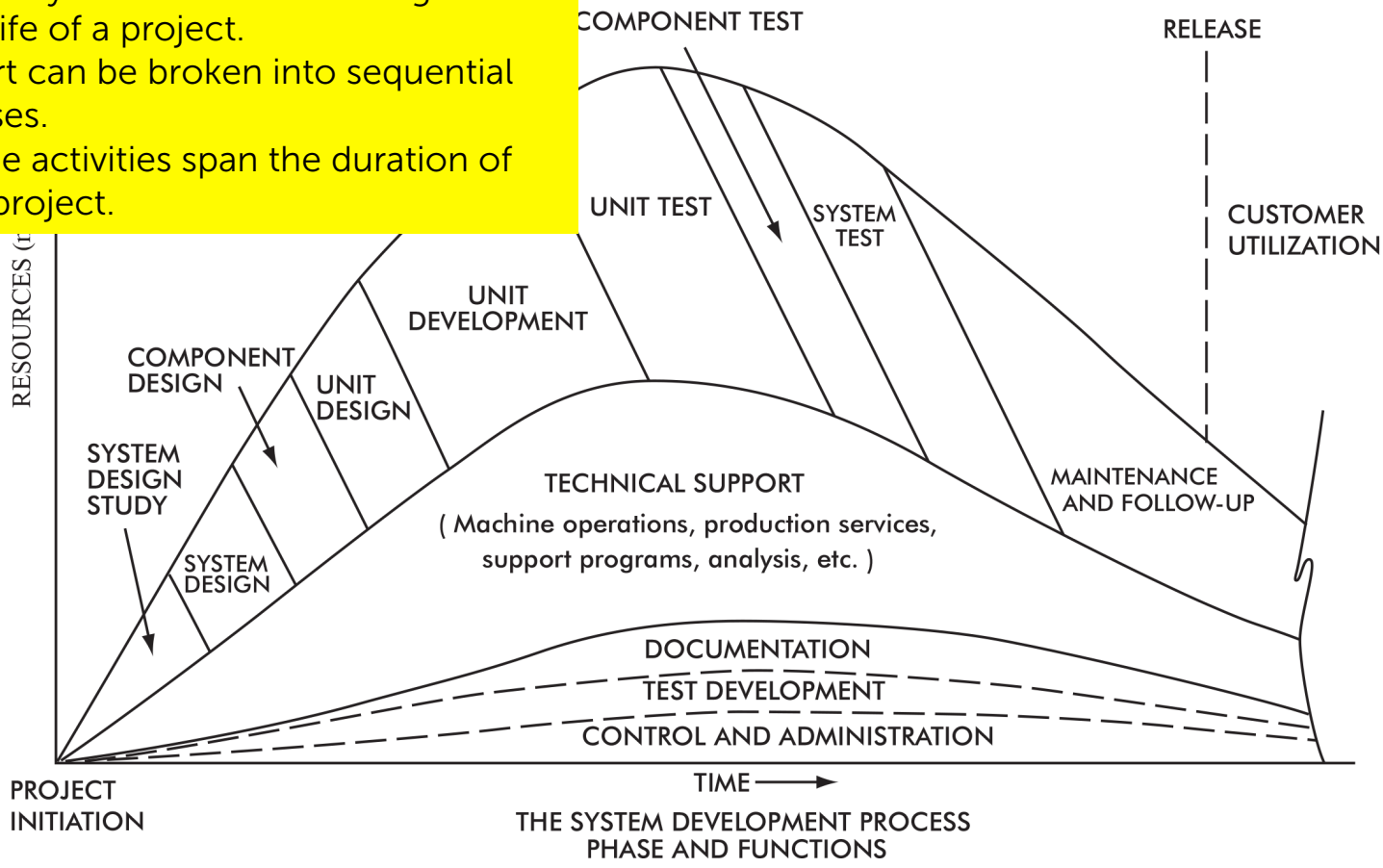

Unlikely chemistry? - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Links Google Java 2 Docs UVic CSC350 Sprin
Address http://www.blonnet.com/businessline/2001/03
Failure record
Done
Internet
Failure record
Companies in the US spend over $250 billion each year on IT application development of approximately 175,000 projects. The average cost of a development project for a large company is $2.3 million; for a medium company, it is $1.3 million; and for a small company, it is $434,000. A great many of these projects will fail. Software development projects are in chaos, and we can no longer imitate the three monkeys -- hear no failures, see no failures, speak no failures.
The Standish Group research reveals that a staggering 31.1 per cent of projects will be cancelled before they ever get completed; 52.7 per cent of projects will cost 189 per cent of their original estimates. The cost of these failures and overruns is just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in trillions of dollars. One just has to look to the City of Denver to realise the extent of this problem. The failure to produce reliable software to handle baggage at the new Denver airport is costing the city $1.1 million per day.
Risk is always a factor when pushing the technology envelope, but many of these projects were as mundane as a driver's licence database, a new accounting package, or an order entry system.

Software Engineering: context

- "Software Engineering" as a term was invented in the late 1960s
  - From 1945 to early 1960s, major cost was computing hardware
  - That started to change in the early 1960s
  - Programming environments, languages, and tools were focused on the computer, not the programmer
- By 1967/68 many experts declared a "software crisis". They saw the following:
  - Inability to hire enough trained programmers
  - Cost & budget overruns
  - Buggy software resulting in property damage or theft
  - Software defects leading to injury or even death
- Another view: Programmers were struggling to write code that would be correct, useable, and on time
- Proposal in 1968: To develop and apply principles to the development of software in a manner similar to established engineering disciplines.

Software development effort

1. Intensity of effort varies throughout the life of a project.
2. Effort can be broken into sequential phases.
3. Some activities span the duration of the project.



From NATO report (1968)

Software engineering: definition

- No one definition encompasses all uses of the term
  - Thousands of researchers
  - Tens of thousands of research papers + books
  - Many tools
  - Many disagreements over what problems are most important...
- Wikipedia's 2013 definition will work for our course
  - "The application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software."

45+ years of research

- Since 1968 there have been many advances in software engineering
  - New programming languages
  - Advances in computing hardware
  - Developments in operating systems, networking
  - New computer-based tools supporting software-system construction
  - Much more besides
- As a result:
  - We are now able to develop, deploy and maintain very complex software systems
  - We are better able to manage the construction of such systems
  - We can collaborate on such work while geographically distributed
  - We very often use the computer itself to support the coordination task (e.g., Subversion)

Some areas in software engineering

System Engineering

Requirements Engineering

Analysis Modelling

Design Engineering

Component-Level Design

Architecture Design

User Interface Design

Software Metrics

Software Testing Strategies

Formal Methods

Software Evolution

Re-engineering

Reverse Engineering

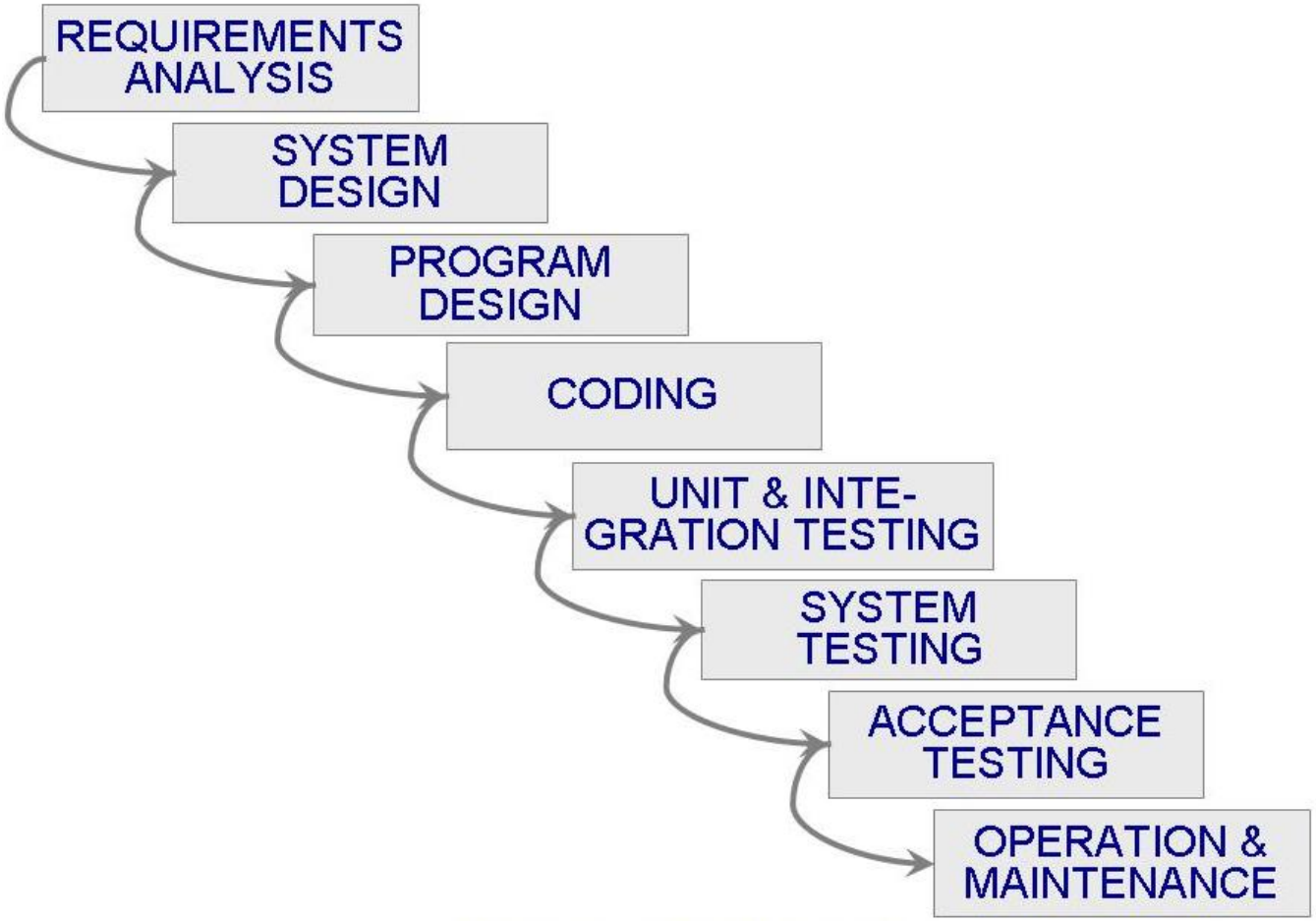
Changing nature of software

- The variety of software systems makes it challenging to describe one single best approach to designing and building software systems
- There are several broad categories of software
  - **System** software
  - **Application** software
  - **Engineering/scientific** software
  - **Embedded** software
  - **Product-line** software
  - **Web applications**
  - **Artificial intelligence (AI)** software
- New challenges in development continue to arise:
  - Open source
  - Ubiquitous computing
  - Cloud computing

Software Process

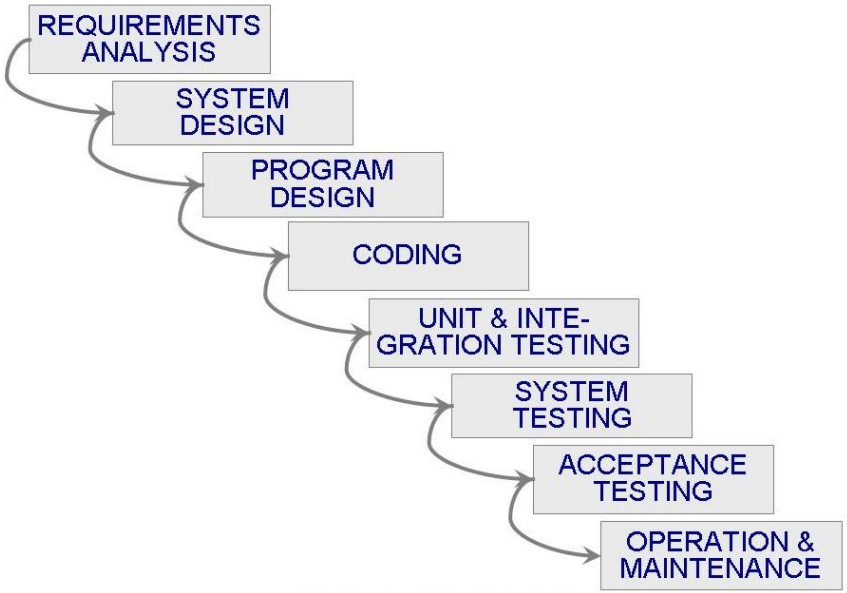
- **Process**
  - A **series of steps** involving **activities**, **constraints**, and **resources** that produce an **intended output** of some kind
  - Involves a **set of tools and techniques**
- Processes are considered important for several reasons:
  - They impose consistency and structure on a set of activities
  - They also guide us to **understand**, **control**, **examine**, and improve the activities
  - Ultimately this enables us to **capture** our experiences and pass them along to future projects

Example process: Waterfall model



Characteristics of a process model

- Prescribes all major process **activities**
- Uses resources, subject to set of constraints (such as a **schedule**)
- Produces **intermediate** and **final products**
- May be composed of **subprocesses** with hierarchy or links
- Each process activity has **entry and exit criteria**
- Activities are organized in **sequence**, so timing is clear
- Each process has **guiding principles**, including **goals of each activity**
- **Constraints** may apply to an activity, resource or product



Why bother modeling a process?

- Phrased differently:
  - "Why don't we stop navel gazing and start writing the darn software right away? We're all pretty smart programmers!"
- Reasons to model a process:
  - To form a **common understanding** amongst team members.
  - To find **inconsistencies, redundancies, omissions** within the process.
  - To **find and evaluate appropriate activities** for reaching process goals.
  - To **tailor a general process** for a particular situation in which it will be used.

An aside: Kinds of coders

- Gandalf
- The Martyr
- Fanboy
- Heavy Metal
- Ninja
- Theoretician
- **Code Cowboy**
- Paratrooper
- Mediocre Wo/Man
- Evangelist

- **Cowboy Coders** are programmers who write code according to their own rules. They may be very good at writing code, but [the code] doesn't generally follow the standards, processes, policies, or anything else derived from the group. Cowboy Coders work well alone, or in the old-style CaveProgrammer environment, but they rarely, if ever, work well in a team. Often times, they are a burr in the saddle that keeps the team from getting positive work done.

http://www.techrepublic.com/blog/10things/10-types-of-programmers-youll-encounter-in-the-field/262

Software life cycle

- Sometimes a **software development process** is also referred to as a **software lifecycle**
- The lifecycle involves some variant and arrangement of these seven phases:
  1. **Requirements** analysis and system **specification**
  2. **System design** (i.e., architecture)
  3. **Program design** (i.e., detailed / procedural)
  4. **Writing** the program (i.e., coding, implementation)
  5. **Testing** (unit testing, integration testing, system testing, acceptance testing)
  6. **System delivery** (i.e., deployment)
  7. **Maintenance**

1. Requirements & Specification

Note the relationship between customer-oriented requirements and team-oriented specifications