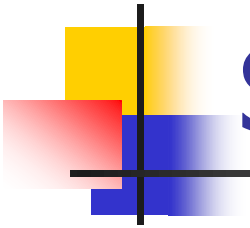
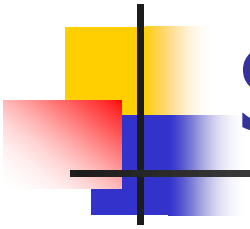


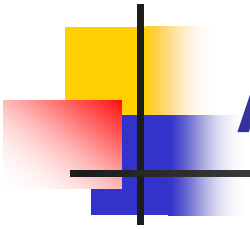
# Software Engineering

- Exciting and frustrating:
- Exciting - Draws on many technical disciplines
  - Provides harness that binds each discipline to the next
  - Frustrating - Demands knowledge in a multitude of topic areas
  - Seems infinitely expandable



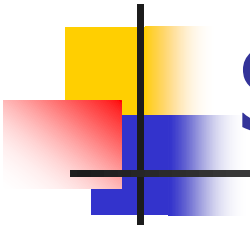
# S. E. Principles

- First proposed in the 1960s
  - One thing remains constant:
    - The desire to apply an engineering discipline to the creation of computer software.
- To understand software engineering:
  - You must develop a consistent view of computer software.
  - Most people think they know what it is
  - Many take a view that’s far too limited



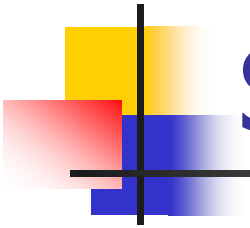
# A Brief Exercise

- **Question**: What is Software?
  - Software is composed of at least three things. List them.



# Software

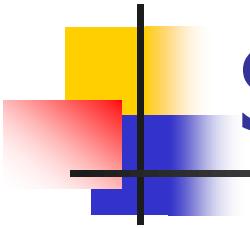
- In making your list you've taken your first step towards understanding how to develop and/or manage computer software
- Many who develop or manage software daily still can't complete the above exercise successfully.



# Software

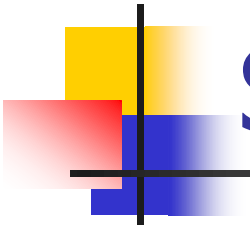
- Fuel powering us in the 21st century, but dangerous if used unwisely
- Fuel that has ignited vast repository of government and corporate info

  that can be tapped to provide insight for political and business leaders.
- The power behind intelligent products that provide features perceived to be science fiction years ago.



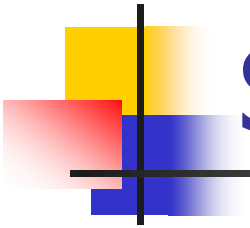
# Software

- The propellant for computer-based systems that assists scientists and engineers in their effort to design and manufacture more advanced products and processes.



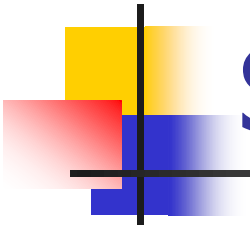
# Software Engineering

- The energy behind computer-based education - K to college level
- Transported the computer into homes and led to computer-assisted banking, shopping, entertainment, info management etc.
- It's the energy that powers the computer revolution --- This makes some people uneasy. Why?
- Opportunities offered by software are accompanied by dangers:



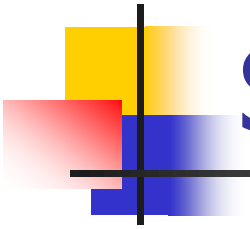
# Software Engineering

- Day-to-day activities of industrialized nations are inextricably linked to software.
- Software failure?
  - no telephone service
  - no printing of paychecks
  - car won't start
  - design error may be introduced into a multi-billion dollar satellite etc.
- A lot has been learnt about how to build high-quality software but we still continue to struggle to meet current needs.



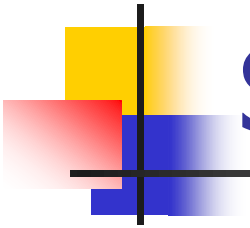
# Software Engineering

- Computer software leverages info technology, but also has the potential to retard its growth.
- **But**
  - Why is this any concern of yours?
  - Aren't the dangers and opportunities associated with computer software the concern of data processing professionals?
- They build the systems, yes, but ….



# Software Engineering

- You use these systems and products
- You rely on the info they create
- You're affected by decisions made as a consequence of the info generated
- You're inconvenienced when the software goes awry.
  - See Alvin Toeffler's book on Rapid Technology. Be prepared to discuss!!

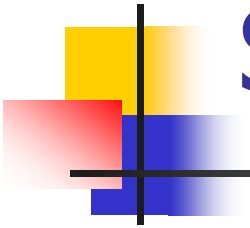


# Software Engineering

- Software Shock
  - stress and disorientation from software penetration of home, work, product, govt. entertainment, etc.
  - Subtle penetration - invisible
- Demand for change in decision-making approach
- Require understanding technology that's foreign to us

# S. E.- **Symptoms of Shock**

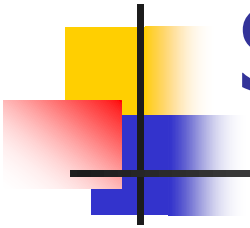
- Technophobia
- Ludits
- Anger
- Unreasonable
- Crisis !!!!



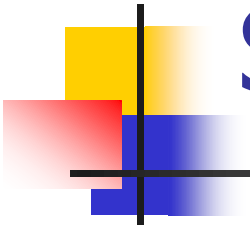
# Software Crisis

- Problems encountered in software development:
  - How we develop software
  - How we maintain existing software
  - Keep pace with growing demand for more software
- Inadequate tools for complexity of solution
  - Labor intensive

# Symptoms of Software Crisis

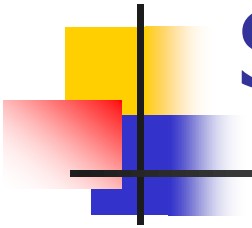


- Software non-responsive to user's needs
- Unreliable
- Excessively expensive
- Untimely
- Inflexible
- Difficult to maintain
- Not reusable



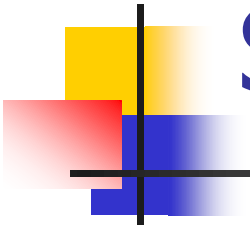
# Software Crisis

- Recently - insight into complexities of large systems and how to manage them



# Solution to Underlying Problems

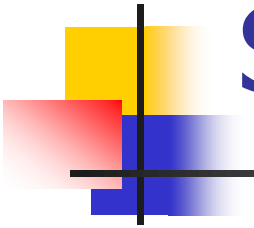
- Need for disciplined approach to software development ..
- Proper vehicle for expressing and executing designs ...
- Problem !! Languages existing prior to massive software-intensive development do not reflect modern s.e principles. To compensate:



# Solution

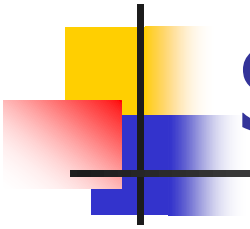
- preprocessors
- extensions
- management controls

- Embedded computer:
- Similar requirements for parallel processing real-time control,high reliability
- Require more than one developer



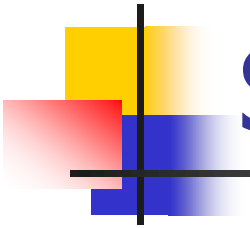
# Solution!!!

- Archaic COBOL/FORTRAN still being used for real-time processing and multi-tasking
- Ada - designed for massive software-intensive systems
  - Major advances in programming technology
  - Embodies software engineering principles
  - Enforces programming practices
- CASE Tools
  - UML



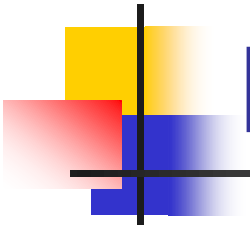
# Software Engineering

- A systematic approach to the creation and ownership of software
- Discipline emerged because developers in the 60s and 70s were consistently wrong in their estimates of:
  - Time
  - Effort
  - Cost involved



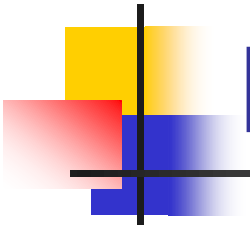
# Software Engineering

- It wasn't uncommon then for systems to:
  1. Cost roughly three times the estimate
  2. Take months, even years, longer than projected
- Reliability and maintainability were serious problems for clients and developers
- Delivered systems:
  - Frequently did not work correctly
  - Effort to fix the problems produced more problems



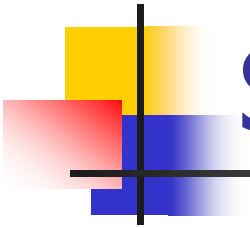
# Reasons for Sorry State

- Managers with no background in software engineering responsible for technical work on major software projects
- Employees with little se experience responsible for difficult technical tasks, such as design or implementation of major portions of software systems, with inadequate technical training and guidance



# Reason for Sorry State

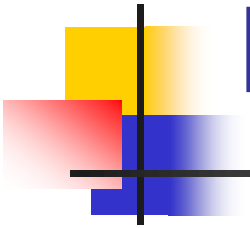
- Graduates of computer science programs at major universities who had never heard of software engineering, let alone the tools and techniques for producing high-quality software products.
- Many computer science programs do not offer courses in software engineering, or if they do, the courses are optional and focus on programming



# S. E. Study

- In our study we'll examine:
  - the product you build
  - the process used to build it effectively
- **The product**
  - ATM, cellular telephones, fax machines, CAT scanners, automobiles, the stack exchange, robots, a Boeing 767 . .
    - **What do they all have in common**?
  - None would work without computer software.

# Project - Getting Started (1)



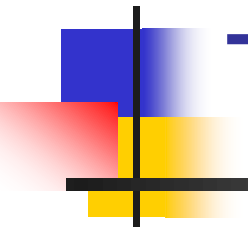
- When? As soon as assigned.
- History of software development:
  - incomplete, unreliable, abandoned projects
- Failures reflect failure to
  - organize, communicate, evaluate
  - take nature of software development into account
  - Failure to apply s.e. principles

# Getting started contd. (2)

- First order?
  - team organization
- Important factors in team organization:
  - team size
  - nature of the project
  - talents of team members

# Team Organization (3)

- Based on two principle ideas:
  - achievement of a goal and division of labor
- Goal cannot be met without group effort or achieve goal more quickly or efficiently
- Key to team success:
  - obtainable goals
  - have clear understanding of goal(s)
  - have wise and workable division of labor



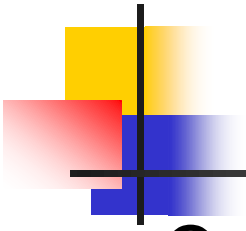
# Team Organization

Team Organization and
*Project Management*

# Brooks' law (1975)

- Adding manpower to a late project only makes it later. *Why?*

- Boehm: A system that has to be delivered too fast gets into the "impossible region"
  - Chance of success becomes almost nil if schedule is pressed too far
  - Why is it useful to explain this reality to project managers?

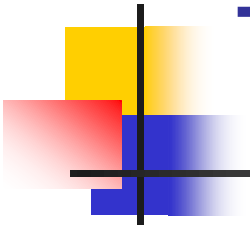


# Brooks' Law revisited

- Quick review: what is Brooks' law?
- "Adding manpower to a late software project makes it later."
- *What does this law (or maxim) imply about the importance of team organization for software development projects?*
- "No substitute for careful planning and team formation if overruns and later confusion, not to mention disaster, are to be avoided."
  -- John S. MacDonald, MacDonald Dettwiler

# Mintzberg's Organizational Configurations

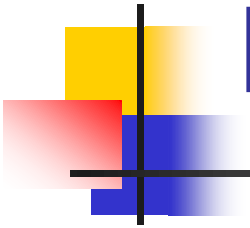
- F ive ways organizations typically configure and coordinate teams:
- **Simple structure** – one or few managers, direct supervision
  - Typically in new, relatively small organizations
- **Machine bureaucracy** – mass-production and assembly lines
- **Divisionalized form** – each division has autonomy
  - Split up work and let each group figure out how to do it
- **Professional bureaucracy** – skilled professionals with autonomy
- **Adhocracy** – for innovative or exploratory projects
- *Which configurations apply for software development projects?*



# Teams (4)

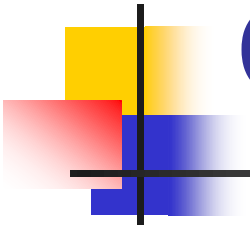
- Democratic:
  - no leader
  - decision by voting or consensus
  - Feasible under limited circumstances
- Formed from peers? can waste time
- Leader expedite matters

# Responsibility of Leader (5)



- Define and set goals
- Establish group organization and delegate responsibilities
- Set deadlines and monitor progress
- Secure necessary resources
- Motivate team
- In industry - chief programmer team

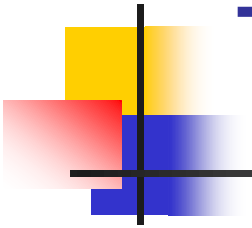
# Chief programmer team (6)



- One chief programmer or leader
  - designs the software
  - implement critical parts
  - makes technical decisions
  - Delegates work
- Besides chief programmer:
- Assistant - liaison with sponsor and groups
  - backup to chief

# Team members (2)

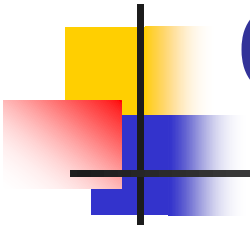




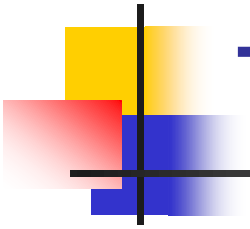
- Librarian - maintains
  - program listings, test plans, test runs
  - handles duties of configuration management
- Tool-smith
  - adapt existing code
  - write customized tools

# Chief-Programmer Structure





- Centralized decision making
- For n-1 programmers there are n-1 communication/decision path instead of $n(\frac{n-1}{2})$
- Technical and managerial skills of chief determines success of the project

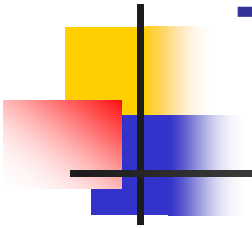


# Team Structure

- Hierarchical team structure -
  - Compromise between the previous two:
  - Project leader
  - Technical leaders
- Egoless programming
  - Alternate to chief programmer
  - Everyone is held accountable
  - Criticism is made of the product/result, not the individuals

# Team members (1)





- Two to five programmers:
  - implement design
  - document source code
  - testing
- Administrator:
  - day-to-day administration detail

# Guidelines for Meetings (1)



- Meetings - central to success of project
  - formal - more or less
  - any topic
- Must have a leader
  - task responsibilities
  - people-oriented responsibilities

# Guidelines for Meetings (2) (11)

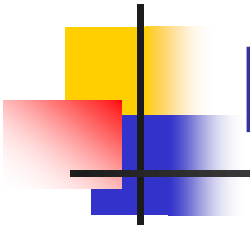
- ensure interaction in a productive, goal-oriented, and motivated way
- Specifically:
  - create agenda
  - determine and notify attendees
  - facilitative role

# Phases of Meeting (12)

- Preparation
  - Decide on purpose and function
- Meeting itself
- Follow-up

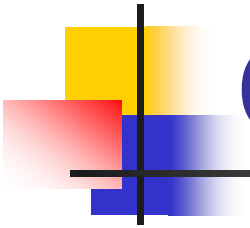
# Possible Functions (13)

- Planning and Preparation
- Policy making
- Managing
- Taking action
- Facilitating action
- Solving problems



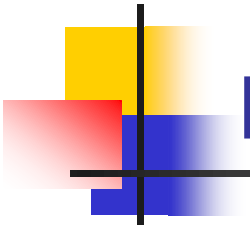
# Requirements for this Class

- You are proficient in a programming language, but you have no or limited experience in analysis or design of a system
- You want to learn more about the technical aspects of analysis and design of complex software systems



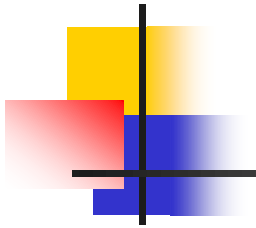
# Objectives of the Class

- Appreciate Software Engineering:
  - Build complex software systems in the context of frequent change
- Understand how to
  - produce a high quality software system within time while dealing with complexity and change
- Acquire technical knowledge (main emphasis)
- Acquire managerial knowledge



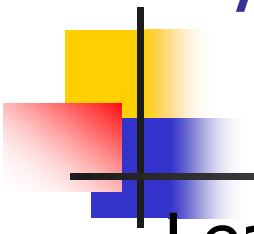
# Focus: Acquire Technical Knowledge

- Understand System Modeling
- Learn UML (Unified Modeling Language)
- Learn different modeling methods:
  - Use Case modeling
  - Object Modeling
  - Dynamic Modeling
  - Issue Modeling



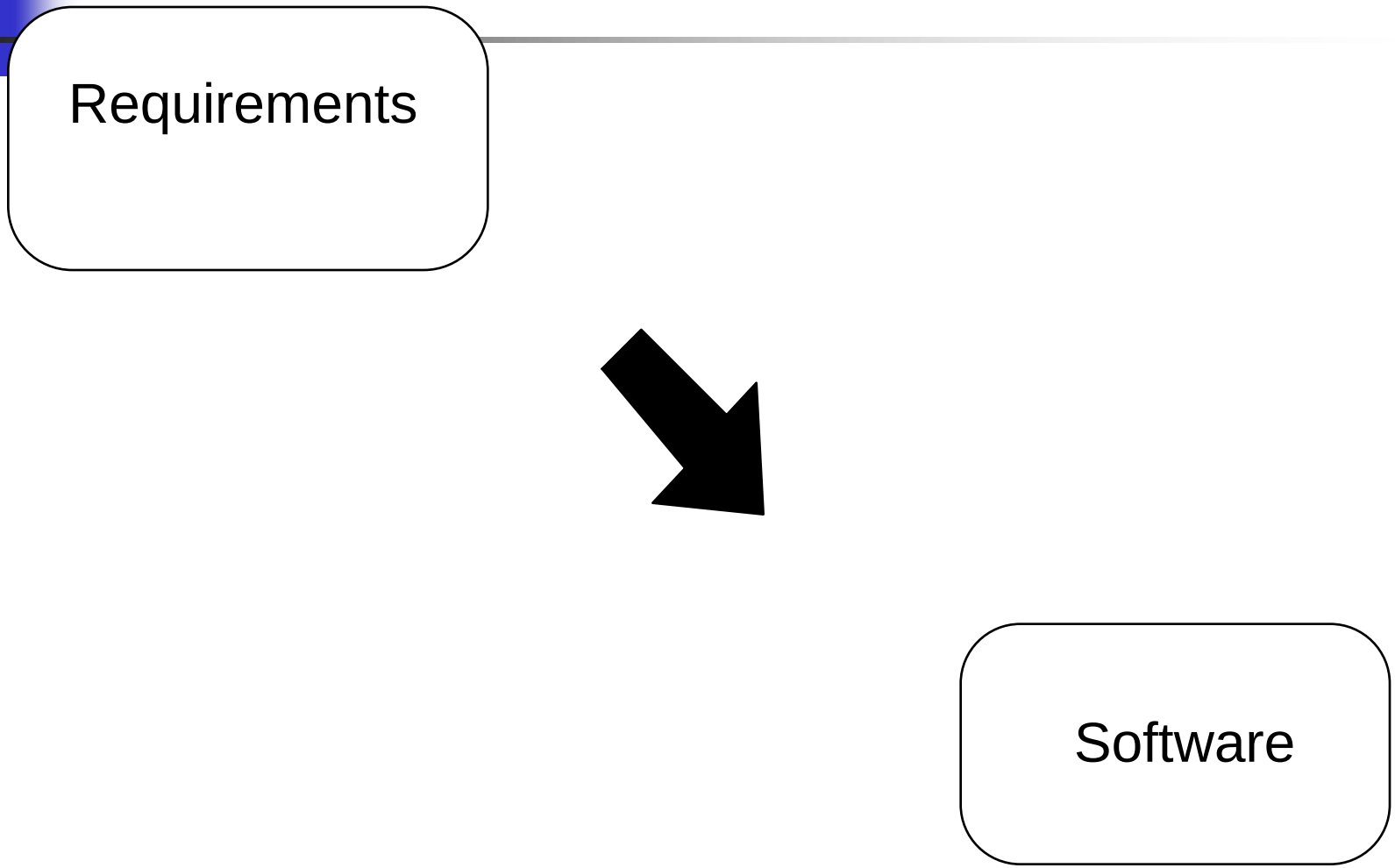
- Learn how to use Tools:
  - CASE
    - Tool: Visual Paradigm (or any other tool of your choice)
- Component-Based Software Engineering
  - Learn how to use Design Patterns and Frameworks

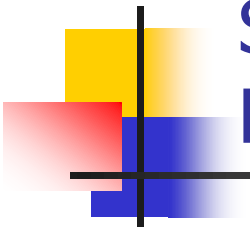
# Acquire Managerial Knowledge



- Learn the basics of software project management
- Understand how to manage a  software lifecycle
- Be able to capture software development knowledge (Rationale Management)
- Manage change: Configuration Management
- Learn the basic methodologies
  - Traditional software development
  - Agile methods.

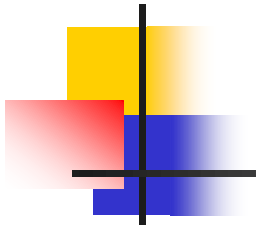
# Limitations of Non-engineered Software





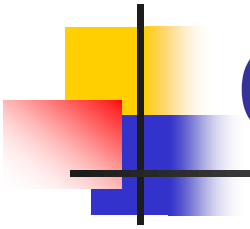
# Software Production has a Poor Track Record Example: Space Shuttle Software

- Cost: $10 Billion, millions of dollars more than planned
- Time:  3 years late
- Quality:  First launch of Columbia was cancelled because of a synchronization problem with the Shuttle's 5 onboard computers:



- Error was traced back to a change made 2 years earlier when a programmer changed a delay factor in an interrupt handler from 50 to 80 milliseconds.
- The likelihood of the error was small enough, that the error caused no harm during thousands of hours of testing.

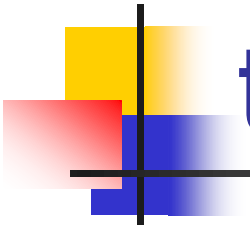
- Substantial errors still exist.
  - Astronauts are supplied with a book of known software problems "Program Notes and Waivers".



# Quality of today's software….

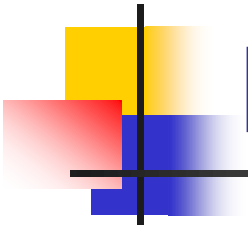
- The average software product released on the market is not error free….
- …has major impact on Users

# Software Engineering is more than writing code



- **Problem solving**
  - Creating a solution
  - Engineering a system based on the solution
- **Modeling**
- **Knowledge acquisition**
- **Rationale management**

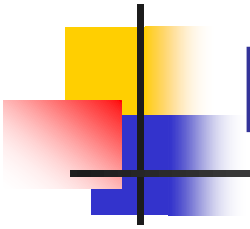
# Software Engineering: A Problem Solving Activity



For problem solving we use

- Techniques (methods):
  - Formal procedures for producing results
- Methodologies:
  - Collection of techniques
- Tools:
  - Instrument or automated systems to accomplish a technique

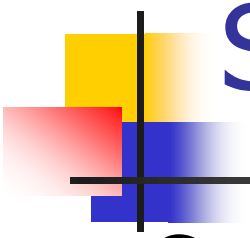
# Software Engineering: Definition



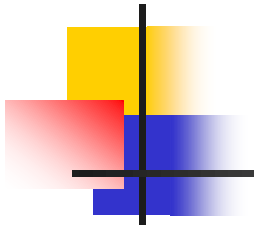
Software Engineering is a collection of techniques, methodologies and tools that help with the production of

- a high quality software system
- with a given budget
- before a given deadline while change occurs.

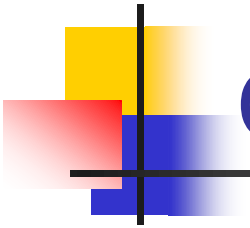
# Scientist vs Engineer



- Computer Scientist
  - Proves theorems about algorithms, designs languages, defines knowledge representation schemes
  - Has infinite time…
- Engineer
  - Develops a solution for an application-specific problem for a client
  - Uses computers & languages, tools, techniques and methods

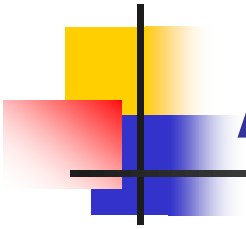


- Software Engineer
  - Works in multiple application domains
  - Has limited time … only 3 months...!!!
  - …while changes occurs in requirements and available technology



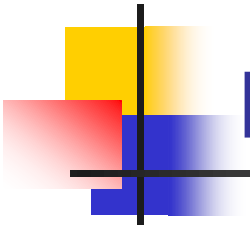
# Objectives of the Lectures

- Appreciate the Fundamentals of Software Engineering:
  - Methodologies
  - Process models
  - Description and modeling techniques
  - System analysis - Requirements engineering
  - System design
  - Implementation: Principles of system development



# Assumptions for this Class

- Assumption:
  - You have taken the prerequisite courses
- Beneficial:
  - Practical experience with a large software system
  - Participate in a large software project
  - You have experienced major problems.



# Focus: Acquire Technical Knowledge

- Different methodologies ("philosophies") to model and develop software systems
- Different modeling notations
- Different modeling methods
- Different software lifecycle models (empirical control models, defined control models)
- Different testing techniques (eg. vertical testing, horizontal testing)
- Rationale  Management
- Release and Configuration Management

# Acquire Managerial Knowledge

- Learn the basics of software project management
- Understand how to manage a  software lifecycle
- Be able to capture software development knowledge (Rationale Management)
- Manage change: Configuration Management
- Learn the basic methodologies, example …
  - Traditional software development
  - Agile methods.