2 midterms (30%); 1 final (20%); class exercises (10%); homework (40%)

class exercises are team based (work with others)

exams closed book/closed notes; seating charts for exams

2 in class exercises dropped

Read chapter 1

Chapter 1: The Scope of Software Engineering

* + Software is:
    - Instructions that when executed provide desired features, function, and performance;
    - Data structures that enable the programs to adequately manipulate information
  + Why is Software different?
    - Software is developed or engineered it is not manufactured in the classical sense
    - Software doesn’t “wear out”
    - Although the industry is moving toward component-based construction, most software continues to be custom-built
  + We don’t wear down software, we destroy it.
  + Draw pic on slide 7
  + Software engineering (IEEE):
    - The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
    - The study of approaches as in (1).
    - Measure what you’re doing
  + Historical aspects
    - 1968 NATO Conference, Garmisch, Germany
    - aim: to solve the software crisis (bad code, no standardization, took too long, couldn’t control it, unexpected outcomes, couldn’t be maintained, $$$)
    - software is delivered
      * late
      * over budget
      * with residual faults
  + draw pic on slide 10
  + cutter consortium data
    - 2002 survey of information tech organizations
      * 78% have been involved in disputes ending in litigation (law suit)
  + conclusion
    - the software crisis has not been solved
    - perhaps it should be called the software depression
      * long duration
      * poor prognosis
  + economic aspects
    - suppose coding method CM\_new is 10% faster than currently used method CM\_old. Should it be used?
    - Common sense answer
      * Yes
    - Software engineering answer
      * Consider the cost of training
      * Consider the impact of introducing a new tech
      * Consider the effect of CM\_new on maintanence
  + Life cycle models
    - Life cycle model
      * The steps to follow when building software
      * A theoretical description of what should be done
    - Import models
      * Waterfall
      * Iterative
      * Incremental
      * Agile
  + Waterfall life cycle model (1970)
    - Requirements phase
    - Analysis phase
    - Design phase
    - Implementation phase
    - Postdelivery maintenance
    - Retirement
    - (issues)
      * requirements change
  + problems ^
    - most of the real projects rarely follow the sequential flow that the model proposes
    - difficult for the customer to state all requirements explicitly before the next step
  + modern maintenance def’n
    - 1995, the internation standards org and international electrotechnical commission defined maintenance operationally
    - maintenance is nowadays defined as dfafdafdsaafs
  + importance of post delivery maint.
    - Development is rare today
    - Reuse is widespread
    - Bad software is discarded
    - Good software is maintained for 10, 20 years or more
    - Software is a model of reality, which is constantly changing
  + Draw pic on slide 19
  + Draw pic on slide 20
  + Consequence of relative costs of phases
    - Return to CM\_old and CM\_new
    - Reducing the the coding cost by 10% yileds at most a 0.85% reduction in total costs
      * Consider the expenses and disruption incurred
    - Reducing postdelivery ….
  + Draw pic on slide 22
  + Requirements, analysis, and design aspects
    - To correct a fault early in the life cycle
    - …
    - between 60 and 70% of all faults in large-scale products are requrements, analysis, and design faults
    - example: jet propulsion lab inspections
      * 1.9 faults per page of specifications
      * 0.9 per page of design
      * 0.3 per page of code
  + conclusion
    - it is vital to improve our requirements, analsy…..
  + team programming
    - hardware is cheap
      * we can build products that are too large to be written by one person in the available time
    - software is built b teams
      * interfacing problems between modules
      * communication problems among…
  + why there is no planning phase
    - we cannot plan at the beginning of the project – we do not yet know exactly what is to be built
  + planning act. In classical paradigm
    - preliminary planning or the requirements and anal…..
  + conclusion
    - planning act. Are carried out throughout the life cycle …
  + why no testing phase
    - do it during development
    - tesing of classical parag.
  + Conclusion
    - Continual testing
    - Testing is responsibility of every developer
    - No testing phase
  + Why no documentation phase
    - Bad bad bad.
    - Too late to document at end
  + Documentation must be current
  + Conclusion
    - Must be ongoing (doc’n)
  + OOP
    - Structured paradigm was successful initially
      * It started to fail with larger products
    - Post …
    - Both data and actions are of equal importance
    - Development is easier
  + Strengths of OOP
    - Can be reused (if good)
* Day 2
  + OOP
    - Both data and action are of equal importance
  + Structured versus OOP
    - Info hiding
    - Responsibility driven design
    - Impact on maintenance dev.
  + Strengths of OOP
    - Well-designed objects are independent units
    - A classical product conceptually consists of a single unit
    - Reuse
  + Structured paradigm
    - There is a jolt between analysis and design
* Chapter 2
  + What is a software life-cycle
    - Software life cycle models allow practitioners to organize software engineering tasks in a certain way.
    - No software process model works well for every project
  + Draw pic ch2 slide 4
  + Choosing and adapting a process
    - Things to consider
      * Flow of activities
      * Required work products
      * Quality requirements
      * Project management
      * Stakeholder involvement
        + Customer, investor, yourself
      * Makeup of team
        + Size, organization
  + Identifying a task set
    - A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
      * What to be accomplished
      * Products to be produced
      * Quality of products
  + Different software life cycle models: classification
    - Plan-driven vs. agile
    - Incremental vs. evolutionary
  + SW Dev in theory
    - Developing linearly, from scratch.
    - Requirements -> analysis -> design -> implementation
  + SW dev in practice
    - We make mistakes
    - Requirements change per cliental
  + Incremental vs. evolutionary
    - Inc
      * Core features first
      * Add other features incrementally
    - Evo
      * Start with small basic idea
      * Gradually add functionality to evolve into fully-fledged software
      * Starting with shell, add stuff as we go
  + Prescriptive life cycle models
    - Advocate orderly approach to SW eng.
  + Waterfall
    - Can apply process to building a house
    - Change is inevitable
    - No solution
  + Millers law
    - At any one time, we can concentrate on only approx. seven chunks (units of info)
    - To handle larger amounts of info, use stepwise refinement
  + Draw pic ch2 slide 19
  + Incremental model
    - Asdf
  + Iterative model
    - Build the shell (if house), then roof, then structure, etc.
  + Workflows
    - Various phases carry heavier weight during parts of cycle
    - (requirements will be heavier at the beginning, rather than the end)
  + strengths of iterative and incremental model
    - multiple opportunities for checking that the sw product is correct
    - robustness of architecture can be determined early on
    - mitigate risks early
    - it works
  + managing iteration and incrementation
    - as regimented as waterfall model
    - incremental dev is a scheduling and staging strategy
    - iterative dev is a rework scheduling strategy