Vad händer i dag?

- Denna föreläsning
  - Kursplan
  - Förra året
  - Årets kurs
  - Pedagogiska tankar
  - Vad är Software Engineering?
- Nästa föreläsning
  - Utvecklingsmodeller
  - Presentation av obligatorisk uppgift

Kursplan

- Kursen ger en översikt över metoder, verktyg och språk för att understödja utvecklingen av stora programvarusystem. Särskild tonvikt läggs på kvalitetssäkring och kunskaper och färdigheter som direkt kan tillämpas i arbetslivet.

Kursplan (forts)

- Moment 1, teoridel, 6 högskolepoäng
  - Utvecklingsmodeller och faser för iterativ och inkrementell utveckling (agile)
  - Kravhantering och objektorienterad analys och design (scenario-driven utveckling, UML)
  - Design heuristics, patterns och refactoring
  - Systematisk testning (test-driven utveckling)
  - Programvarukvalitet, mätetal, processförbättring, versionshantering och dokumentation
- Moment 2, laborationsdel, 9 högskolepoäng
  - Delmomentet utgörs av ett eller flera obligatoriska projekt där olika aspekter från moment 1 tränas och fördjupas. Projekten genomförs i varierande grupper.

Förväntade studieresultat - Kunskap och förståelse

- Efter avslutad kurs ska studenten kunna:
  - visa förståelse för och diskutera kring de tekniska och organisatoriska egenskaper som är förknippade med gruppvis utveckling av stora och komplexa programvarusystem
  - beskriva teorier, modeller och verktyg för att kunna planera, genomföra och analysera utvecklingsprojekt av programvara
  - visa förståelse för hur programvarukvalitet kan förbättras med hjälp av kvalitativa och kvantitativa metoder
  - visa förståelse för hur individen och gruppen påverkar ett projekts resultat

Förväntade studieresultat - Färdighet och förmåga

- Efter avslutad kurs ska studenten kunna:
  - analysera och genomföra processförbättringar inom ett programvaruutvecklingsprojekt
  - följa upp och presentera (muntligt och skriftligt) ett programvaruutvecklingsprojekt
  - systematiskt använda verktyg för modellering, systematisk testning och versionshantering
  - i grupp planera, genomföra och analysera programvaruutvecklingsprojekt

Förväntade studieresultat – Värderingar och förhållningssätt

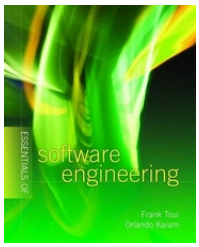
- Efter avslutad kurs ska studenten kunna:
  - visa ett professionellt förhållningssätt inklusive att förstå, styra och utveckla sig själv i syfte att bidra till ett projekts måluppfyllelse
  - bidra positivt till en projektgrupps utveckling

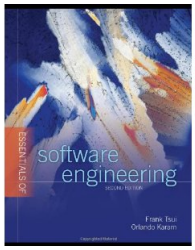
Förra årets kurs

- Allmänt en mycket uppskattad kurs
- En obligatorisk uppgift och ett projekt med flera faser
- Mycket relevant kursinnehåll
- Kursuppläggningen fungerade bra
- Mycket uppskattat med stort ”riktigt” projekt, feedback och Jonas

Årets kurs

- Kursledning
- Kurslitteratur
- Uppläggning
  - Föreläsningar
  - Gästföreläsare
  - En enskild obligatorisk uppgift
  - Ett projekt i två huvuddelar
  - Schema
- Examinering
  - Rapporter, deltentor och projektredovisningar





Obligatorisk uppgift och projekt

- En fristående obligatorisk uppgift
  - Scrum och Lean Software Development
  - Enskilt
- Projektet
  - Ett projekt med två olika delar
  - Olika grupper
  - Projektet genomförs i olika faser
  - Olika typer av redovisningar, även blandat mellan gruppredovisningar och enskilda redovisningar

Examinering

- Många examinerande inslag
  - Nedan visas andelen av slutbetyget
- Tenta – 35%
- Projekt, del 1 – 10%
  - Conceive – 0%
  - Design –10%, varav personlig del 5%
- Enskild obligatorisk uppgift – 15%
- Projekt, del 2 – 40%
  - Sprint 1 – 0%
  - Sprint 2 – 5%
  - Sprint 3 – 10%
  - Sprint 4 – 25%, varav personlig del 5%

Bedömning

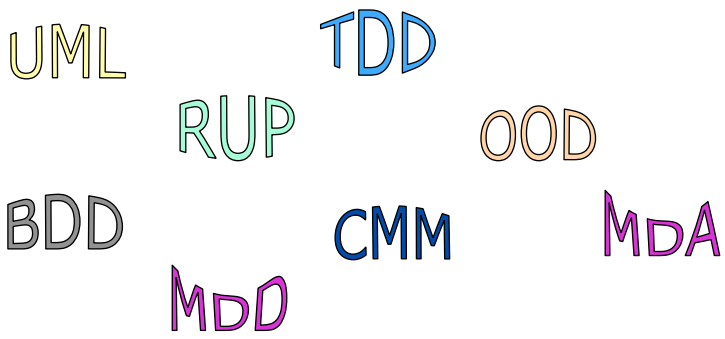
Bedömningsgraderingar	
Grad	**Betydelse**
0	Ej acceptabelt: Inget tecken på lyckat arbete; plagiat; ej levererat; ej dykt upp
1	Svagt: Slarvigt arbete; saknar betydande delar/aspekter; endast mindre delar uppfyller presentationens eller arbetets syfte(n)
2	Under medel: Saknar några delar/aspekter, men de viktigaste finns med; uppfyller inte till fullo syfte(n)
3	Medel/acceptabelt: I stort uppfylls syfte(n); saknar endast mindre viktiga delar/aspekter; uppfyller lägsta kraven för tillfredställande prestation
4	Över medel: Uppfyller syfte(n) väl; uppfyller mer än lägsta kraven i de flesta delar/aspekter; en känsla av professionalitet
5	Väldigt bra: Uppfyller mycket mer än lägsta kraven i nästan alla delar/ aspekter; professionellt
6	Outstanding: Över förväntan för vanliga studentprojekt i nästan alla delar/aspeketer

Pedagogik

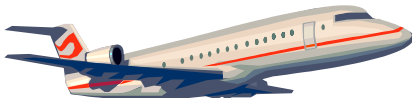
- Pedagogisk tanke
- Val av presentationshjälpmedel
- OH-bilder
  - Källa
  - Innehåll
  - Språk
- Vad bör man lära sig?

Software Engineering: Why, What, and How ...

- ... developing large systems is different
- ... and a lot of three-letter acronyms



Why Do We Need SE?


- Software is everywhere
- Software becomes more and more complex
- Software failures may do harm
- Software projects exceed budgets and schedules
- ...

➡ Software must be engineered like any other product

Classical Software Failures

- Therac-25 radiation overdoses (erroneous non-critical modules of previous Therac-20 software were reused in a safety-critical role)
- Ariane 5 launch failure (specification of the reused modules from Ariane 4 software was misunderstood)
- Loss of the Mars Climate Orbiter (navigation error due to incompatible measurement units)
- FBI's virtual case file system (scope creep, bad management)
- ...
- Check Risks Digest for more (and the details)

What is Software Engineering 1?

"The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."

NATO conference '68 in Garmisch

" (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
(2) The study of approaches as in (1)."

IEEE std 610.12-1990, 1990; adopted in SWEBOK

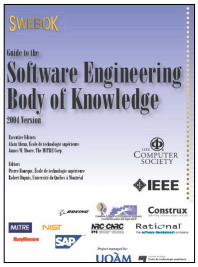
"A broad field that touches upon all aspects of developing and supporting a software system:

1. Technical and business processes
2. Specific methodologies and techniques
3. Product characterization and metrics
4. People, skills and team work
5. Tools and training
6. Project coordination and management"

Tsui and Karam, 2011

What is Software Engineering 2?

- SWEBOK Knowledge Areas
  - Software requirements
  - Software design
  - Software construction
  - Software testing
  - Software maintenance
  - Software quality
  - Software configuration management
  - Software engineering management
  - Software engineering process
  - Software engineering tools and methods
- SWEBOK Related Disciplines
  - Computer engineering
  - Computer science
  - Management
  - Mathematics
  - Project management
  - Quality management
  - Software ergonomics
  - Systems engineering



To Engineer Software Means ...

- Fighting complexity and uncertainty
- Systematic, disciplined, and quantifiable work
- Based on proven principles and practices
- Professional practice

SE Principles and Practice 1

- Apply and use quantitative measurements in decision-making
- Build with and for reuse
- Deal with different individual aspects of the problem by concentrating on each one separately
- Define software artefacts rigorously
- Establish a software process that provides flexibility
- Implement a disciplined process and improve it continuously
- Invest in the understanding of the problem
- Manage quality throughout the life cycle as formally as possible

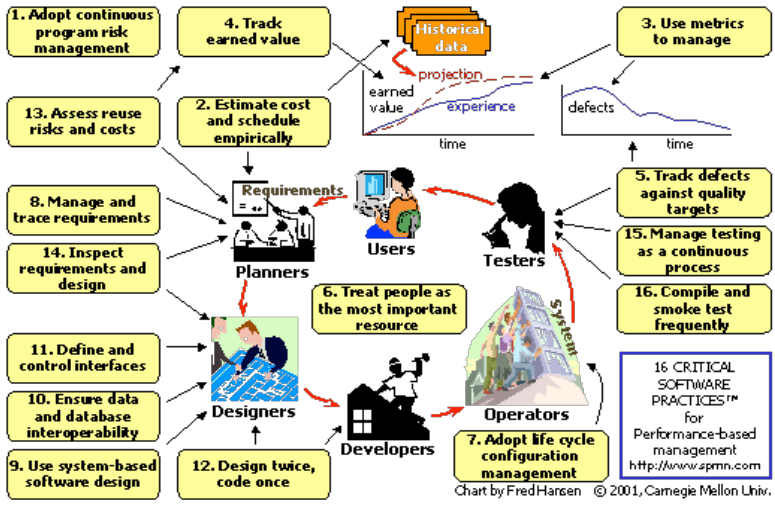
Journal of Systems and Software ***62*** (1)

SE Principles and Practice 2

- Minimize software components interaction
- Produce software in a step-wise fashion
- Set quality objectives for each deliverable product
- Since change is inherent to software, plan for it and manage it
- Since tradeoffs are inherent to software engineering, make them explicit and document them
- The requirements must be firm and fixed
- The tools, methods, and support systems must be designed and selected to support the software engineers
- Uncertainty is unavoidable in software engineering. Identify and manage it

Journal of Systems and Software ***62*** (1)

16 Critical Software Practices™



Software's Ten Essentials

- A product specification
- A detailed user interface prototype
- A realistic schedule
- Explicit priorities
- Active risk management
- A quality assurance plan
- Detailed activity lists
- Software configuration management
- Software architecture
- An integration plan

IEEE Software **14**(2), Mar/Apr 1997, 143-144

Software Engineering Code of Ethics and Professional Practice

1. **PUBLIC** - Software engineers shall act consistently with the public interest.
2. **CLIENT AND EMPLOYER** - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. **PRODUCT** - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. **JUDGMENT** - Software engineers shall maintain integrity and independence in their professional judgment.
5. **MANAGEMENT** - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. **PROFESSION** - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. **COLLEAGUES** - Software engineers shall be fair to and supportive of their colleagues.
8. **SELF** - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

ACM/IEEE-CS (V 5.2)

A Question ...

How could that have helped in the classical failures described before?

Discussion—The Fossbakk Case

- Norwegian customer lost ~ $100,000
- Internet banking interface accepted only 11 digits for account numbers
- Superfluous digits are deleted
- Fossbakk inserted 715815555022 instead of 71581555022 (i.e., an extra '5')
- The UI deleted last digit ⇒ 71581555502
- **Problem**: This actually was a valid account number

(In a similar case a bank stripped "erroneous" commas from the amount to pay, i.e., $1000,50 (thousand dollar and 50 cents) became $100050)

Who is responsible?

Source: IEEE Computer, Apr 2008 and Jun 2008, respectively

The Message

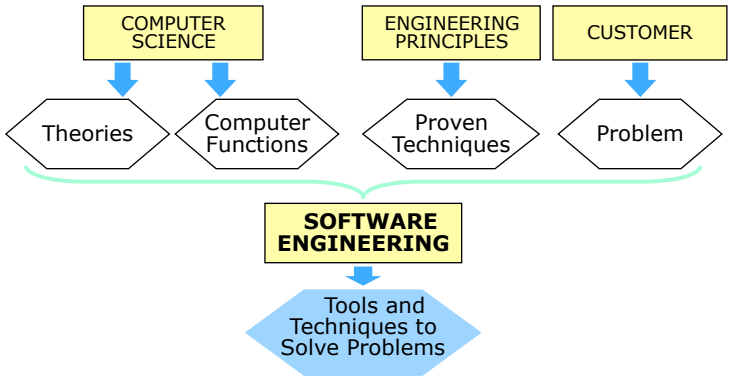
Software Engineering is
software construction
with a big 'E'

Elements of Software Engineering

- Methods
  "How to's" to support software development tasks
- Languages
  Notations to support methods
- Tools
  Support for (the usage of) methods and languages
- Processes
  Coordination and management of software development tasks supported by methods, languages, and tools

➡ **Economically** produce **quality** software

What is Software Engineering 3?



Davis's Early 15 principles (1994)

1. ***Make quality number 1***
2. ***High quality software is possible***
3. ***Give products to customers early***
4. ***Determine the problem before writing the requirements***
5. ***Evaluate design alternatives***
6. ***Use an appropriate process model***
7. ***Use different languages for different phases***
8. ***Minimize intellectual distances***
9. ***Put techniques before tools***
10. ***Get it right before you make it faster***
11. ***Inspect code***
12. ***Good management is more important than good technology***
13. ***People are the key to success***
14. ***Follow with care***
15. ***Take responsibility***

Are these consistent within themselves - - -
How "key" is "people are the key to success"
What do you think?

Royce's More Modern Principles (1998)

1. *Base the process on an architecture first approach*
2. *Establish iterative process --- address risk early*
3. *Emphasize component based development to reduce effort*
4. *Establish change management*
5. *Use round-trip engineering – a form of iterative process*
6. *Use model-based and machine processable notations for design*
7. *Establish process for quality control and project assessment*
8. *Use approach that allows artifacts to be demonstrated early*
9. *Plan to have incremental releases*
10. *Establish a configurable process to suit the needs*

Agree with these? Why ?

Wasserman's Fundamental Concepts (1996)

1. *Abstraction*
2. *Analysis and design methods and notation*
3. *User interface prototyping*
4. *Modularity and architecture*
5. *Reuse*
6. *Life cycle and process*
7. *Metrics*
8. *Tools and integrated environment*

Important concepts - - - how do they relate to earlier listed principles from Davis or Royce?

How about principles for Support?

- Is there a need to come up with some principles or rules for supporting and maintaining released software? What about:
  - customer is always right
  - problem resolution turn around time is key
  - problem fix or resolution quality is vital

Any others? Do resolution speed and quality conflict with each other?

Sammanfattning

- Kursplan
- Förra året
- Årets kurs
- Pedagogiska tankar
- Vad är Software Engineering?

Resten av dagen

- Nästa föreläsning
  - Utvecklingsmodeller
  - Presentation av obligatorisk uppgift