CSc 131
Computer Software Engineering

Chapter 1
SWE Profession

Herbert G. Mayer, CSU CSC

Status 9/4/2019

Syllabus

- Objectives
- History
- SWE Profession
- SW Development Process
- SW Crisis
- Code of Ethics
- Summary
- References

Objectives

Objectives

- Introduce software engineering (SWE) and explain its importance in the world of engineering with computers and computing
- Propose answers to key questions about software engineering
- Introduce ethical and professional issues and explain why they are of particular concern to software engineers

Objectives

- Economies of ALL developed nations are dependent on software
- More and more systems are software controlled
- *Software Engineering* is concerned with theories, methods and tools for professional software development
- Software engineering expenditure represents a significant fraction of GNP in all developed countries

FAQ About SWE

Critical points, before diving deeply into SWE:

1. **SW costs often dominate system costs**
2. **Software cost on digital computers is often higher than hardware cost**
3. **SW costs more to maintain than it does to develop. For systems with a *long life*, maintenance costs may be several times development costs**
4. **SWE is concerned with cost-effective development of such a high-priced, high-value commodity; commodity being that SW product!**

FAQ About SWE

Critical points, before diving deeply into SWE:

5. **What is software?**
6. **What is software engineering (SWE)?**
7. **What is the difference between software engineering and computer science?**
8. **What is the difference between software engineering and system engineering?**
9. **What is a software process?**
10. **What is a software process model?**

General FAQ About SWE

- What are costs associated with software engineering?
- What are software engineering methods?
- What is CASE (Computer-Aided Software Engineering)?
- What are attributes of good software?
- What are key challenges facing software engineering?

Are Women Active in SWE?

What is SW Engineering?

Intuitive, non-scientific definition of SWE:

- **An engineering discipline of computer science . . .**
- **That produces correctly working, efficient, low-cost SW solutions to defined compute problems . . .**
- **In a way that achieves defined SW engineering goals**
- **Key goals for any SWE product depend on project; they include:**
  - **Correct function; that which works, must work correctly**
  - **Complete function; all "promised features" must work**
  - **Acceptable, defined speed; must be defined**
  - **Low cost of development and maintenance**
  - **Portability to other compute environments**
  - **Intuitive, simple user interface; developer finds it easy!**
  - **Complete documentation; common tools, not special env.**
  - **High stability and reliability, etc.**

What is SW Engineering?

From Wikipedia [1] four literal quotes about SWE:

- “The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software” – The Bureau of Labor Statistics – IEEE Systems and SW engineering
- "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software” – IEEE Standard Glossary of Software Engineering Terminology
- “An engineering discipline that is concerned with all aspects of software production” – Ian Sommerville
- “The establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines” – Fritz Bauer

What is SW Engineering?

- Isn't this obvious?
- Yes, just like advice to succeed in the stock market: "Buy low, sell high!"
- Just like common advice for leading a righteous and successful human life: "Always do the right thing!"
- Yes, obvious! But equally hard to do and to completely follow and grasp! ☺
- Like my manager at Burroughs many years ago advised me, building a new compiler for the company's next generation computer: It is simple:
  - Building a compiler? Its is easy Herb!
  - Just build the FE
  - Then the middle-end
  - And then complete the BE!. Bingo!

What is SW Engineering?

- **Software engineering is a computer-based engineering discipline which is concerned with all aspects of software production**
  - **Goal of SW product, funding for productization, time, plan, staff, design, document, implement, test, correct and verify, release, publish, productize, update, EOL**
- **Software engineers should adopt a systematic and organized approach to their work;** else they won't be SWE long-term anyway ☹
- **And use appropriate tools and techniques depending on the problems to be solved, and on the development constraints and resources available**

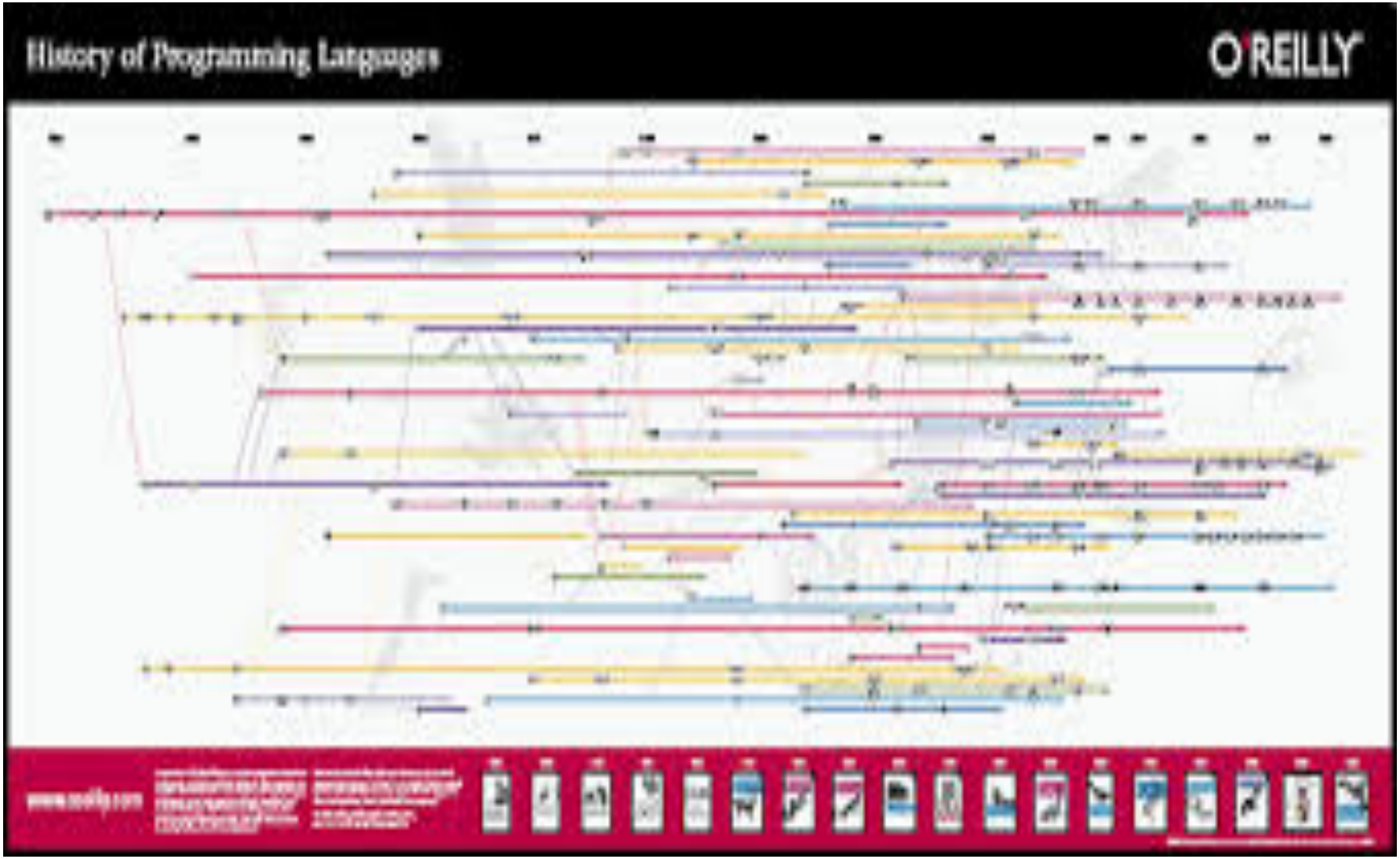
History

of Computers and Programming Languages

History

- Early computers 1940s, had hard-wired instructions
- Were easy to use, but inflexible! Single use!
- Instead "stored program" architecture was needed, resulting in von Neumann computer design
- Programming languages appeared in 1950s, e.g. assembly, Fortran, Cobol; trend continued with many additional languages into 1960s; luckily fewer languages nowadays!
- David Parnas argued 1960s for modularity, for information hiding 1970s [2]
- Term "Software Engineering" evolved 1960s, popularly documented by Bauer [3]

History of Programming Languages

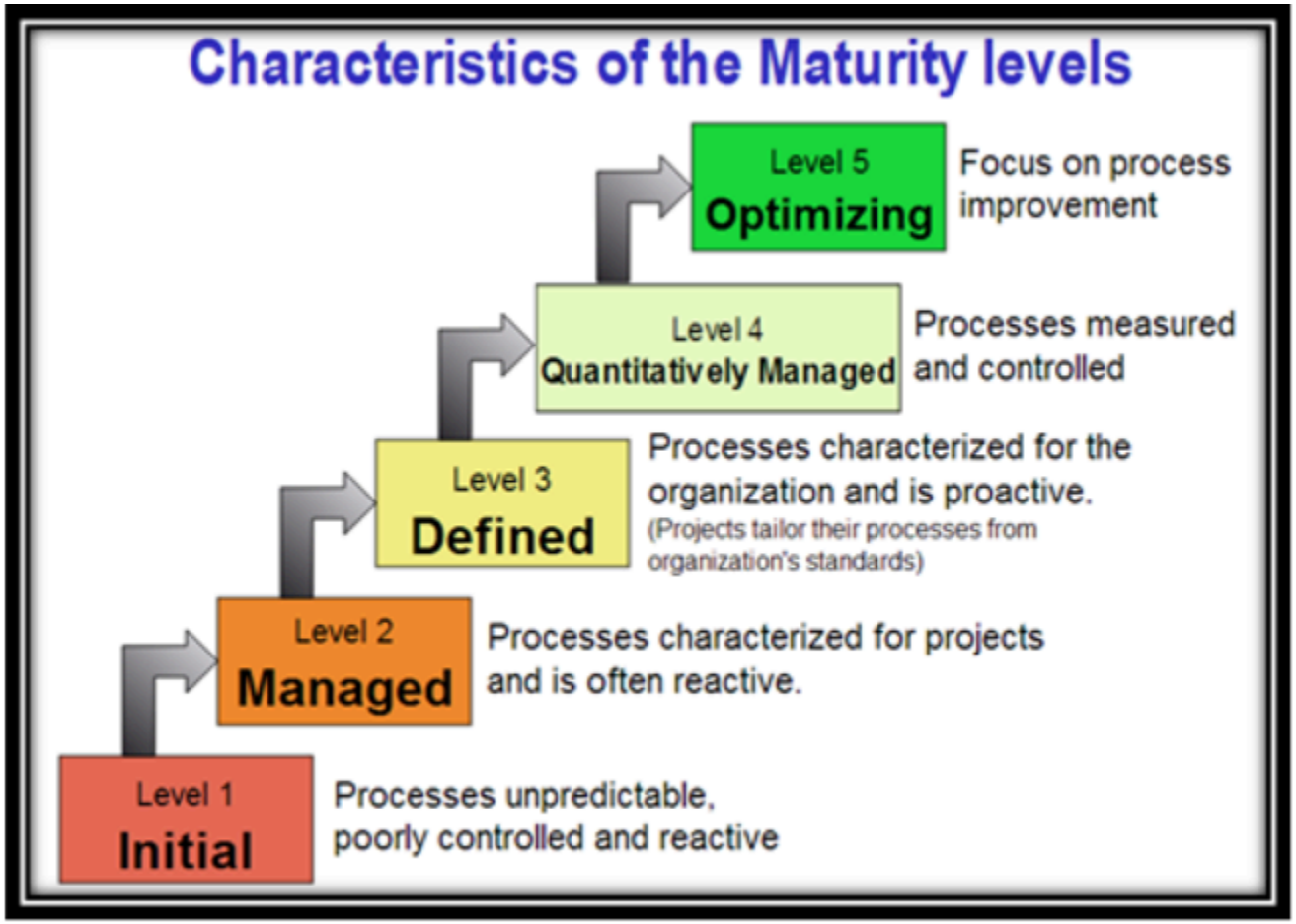


History

- Software Engineering (SWE) institute established, federally funded, 1984 on CMU's campus
- Watts Humphreys founded and led Carnegie Mellon SWE Institute (SEI) in 1986
- Humphreys [4] AKA "Father of SW Quality"!
- Aimed at understanding, managing SWE process
- Process Maturity Level was introduced, popularized
- Morphed into Capability Maturity Model, for SW development process; aka CMM

History

- CMM used to date by US government to evaluate maturity level of government funded SW projects
- CMM used in mature SW industry (so rare! ☹)



SWE Disciplines

Various sub disciplines of SWE evolved; among others:

- SW requirements specification: elicitation, analysis, specification, and validation of requirements for SW
- SW design: process of defining architecture, components, interfaces, other characteristics of system or component
- SW testing: empirical, technical investigation conducted to provide stakeholders with information about product quality
- SW maintenance: totality of activities required to provide cost-effective support to software

SWE Disciplines

Sub disciplines, cont'd:

- **SW development process: definition, implementation, assessment, measurement, management, change, and improvement of SW life cycle process**
- **SWE models and methods: impose structure on software engineering; goal to render activity systematic, repeatable, and ultimately more success-oriented**
- **SWE professional practice: treats knowledge, skills, and attitudes that SW engineers must possess to practice SWE in a professional, responsible, and ethical manner**
- **SWE economics: making decisions related to software engineering in a business context**

SWE Profession

SWE Profession

- Requirements for licensing or certification of professional SW engineers vary around the world
- UK for example has no licensing or legal requirement to assume title of SW engineer
- In Europe, SW Engineers can obtain the European Engineer professional title (EUR Ing.)
- In 2013 US has offered NCEES professional engineer exam; allows SWE licensing and thus recognition
- Mandatory licensing (was being) hotly discussed in the US
- US and UK had initiated formal SWE certification programs; found relatively little interest

SW Development Process

- In SWE, the SW development process is method of dividing SW development into distinct phases:
  - SW design
  - Project management, and
  - Product maintenance
- Commonly known as SW Development Life Cycle
- More recent processes are characterized as agile
- Other common methodologies are:
  - Waterfall methodology
  - Prototyping (or fast prototyping)
  - Iterative and incremental development
  - Spiral development
  - Rapid application development
  - Extreme programming

SW Development Process

- Some process methodologies include specific prescriptive steps, directing organizations day-to-day
- Others are flexible and let SW organization decide
- History, 1970s
  - Structured Programming (USA)
  - Cap Gemini System Development Methodology (France)
- 1980s
  - Structured system analysis and design method (SSADM)
  - Information requirement analysis
- 1990s
  - Object-oriented programming (OOP), dominant 1995
  - Rapid application development (RAD)

SW Development Process

- 1990s Cont'd
  - Dynamic systems development method (DSDM)
  - SCRUM, pushed by Mountain Goat SW [5] since 1995
  - Team SW process, since 1998
  - Rational Unified Process (RUP), by IBM since 1998
  - Extreme programming, since 1998
- 2000s
  - Agile unified process (AUP), ~2005 Scott Ambler
  - Disciplined agile delivery (DAD), superseded AUP
- 2010s
  - Scaled Agile Framework (SAFe)
  - Large-Scale SCRUM (LeSS)
  - DevOps

SWE Profession

- Goal of SWE profession is to *successfully* manage large SW projects
- Managing means: proposing, "selling" for funding, planning, implementing, testing, validating, updating and ending (EOL) large products that are SW based
- Same management principles should be used in small and medium SW product development cycles
- Yet violating, omitting sound principles receives *smaller punishments* ☺ when the project is not large
- *Large* not rigorously defined; guideline: a SW product *way over 100k LOC* can be viewed as large
- Typically created by a team, not a single engineer
- Needs even more planning than described here

Attributes of Good SW

- Software product must produce required functionality and performance to end-user, should be maintainable, dependable and usable
- Maintainability
  - Software must evolve to meet changing needs
- Dependability
  - Software must be highly trustworthy ; predictable
- Efficiency
  - Software should not make wasteful use of system resources
- Usability
  - Software must be satisfactory to end user; AKA ease of use

SWE Challenges

- Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times
- Legacy systems
  - Old, valuable systems must be maintained and updated
- Heterogeneity
  - Systems are distributed and include a mix of hardware and software
- Delivery Time
  - There is increasing pressure for faster delivery of software

Legacy Systems?

SWE Crisis

SWE Crisis

- Software crisis: Happened early during explosive evolution of computer industry due to difficulty of writing useful, efficient programs in required time
- Crisis was due to rapid increase in computer power and problem complexity that could not adequately be tackled at the time
- With increase in SW complexity new SW problems arose because existing methods were insufficient
- Term “Software Crisis” (among others) coined by attendees at first NATO SWA Engineering Conference in 1968 in Garmisch Partenkirchen, Germany; see below
- Edsger Dijkstra at his 1972 Turin Award ceremony coined the same term Software Crisis!

SWE Crisis

Causes of crisis linked to complexity of HW and SW

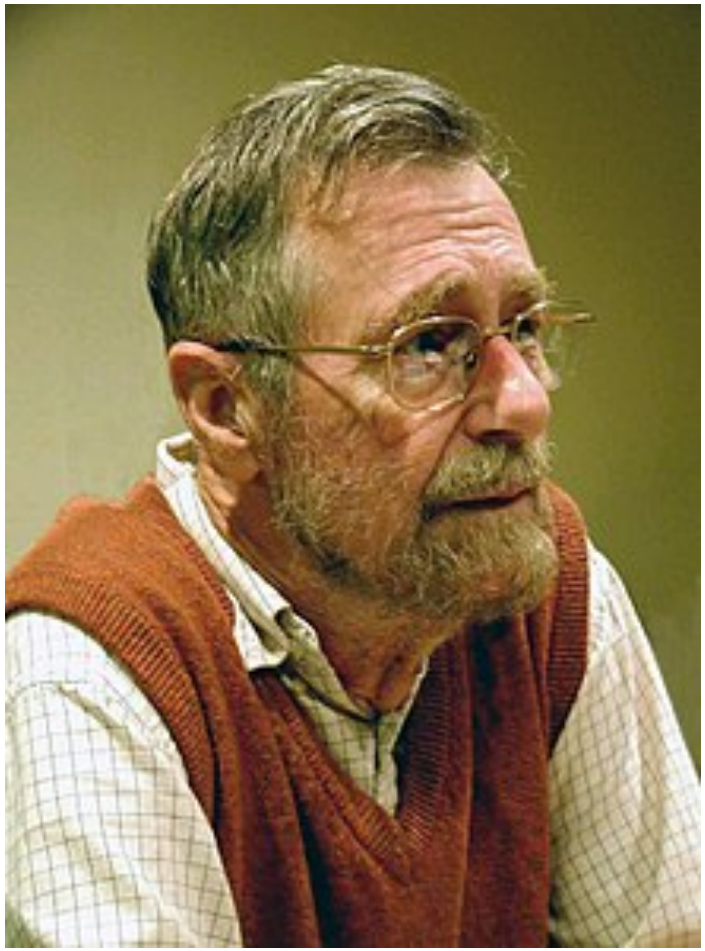
- Projects running over-budget
- Projects running over-time
- SW ran inefficiently
- Software generated was of low quality (meaning?)
- Live Software failed to meet requirements
- Projects were unmanageable, code hard to maintain
- Some SW was never delivered!

Dijkstra quote regarding SWE crisis:

"The major cause of the software crisis is that the machines have become an equally gigantic problem."

See detail below

Edsger Wybe Dijkstra

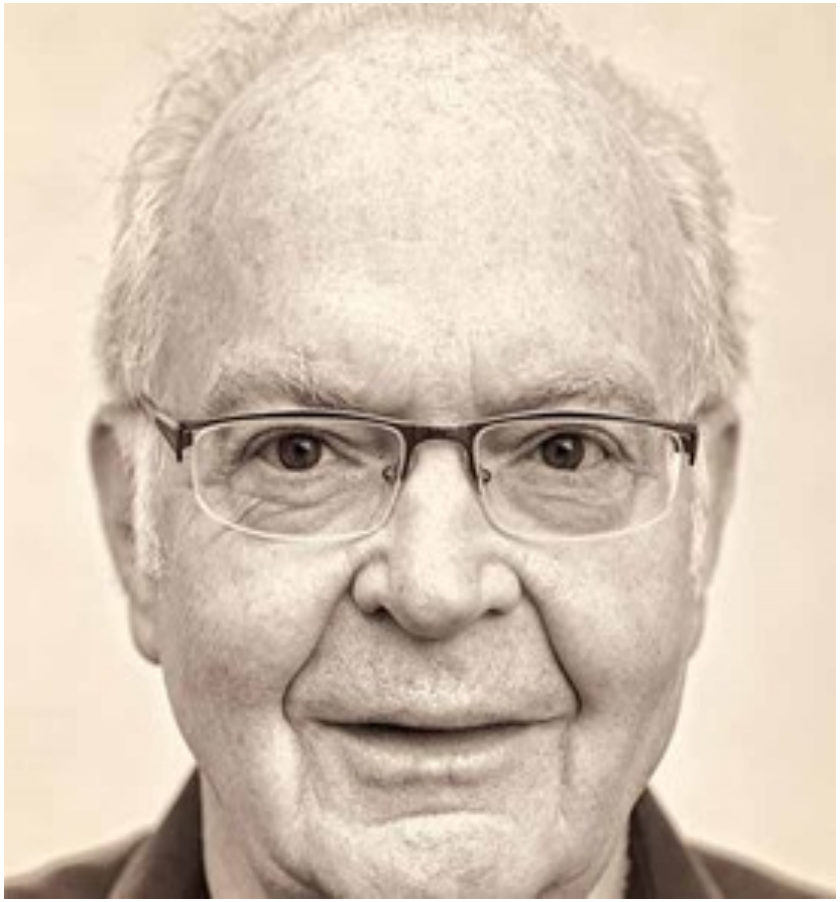


The Humble Programmer. 1930 - 2002

SWE Contributor Dijkstra

- Known for defining the "Shortest Path First Algorithm" to search in graph for path from node a to node b
- Articulated ominous term Software Crisis:
  - "The major cause of the software crisis is that the machines have become several orders of magnitude more powerful!
  - To put it quite bluntly: as long as there were no machines, programming was no problem at all
  - When we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."
- Term "Software Crisis" discussed again by Dijkstra in his acceptance speech at 1972 Turing Award

Donald Ervin Knuth



Donald E. Knuth, “The Art of Computer Programming”

SWE Contributor Knuth

- Known for “The Art of Computer Programming”, colloquially ☺ AKA: TAOCP
- ***Must read*** material for CS students & professionals
- Started out as a single book with 12 chapters
- Grew into multi-volume project
- Of which 4 volumes have been fully completed
- Last volumes estimated to be completed late this year! See [6]
- Recipient of 1974 Turing Award
- Creator of TeX typesetting system
- Enthusiastic organ player

Tony Hoare

Quicksort Inventor

SWE Contributor

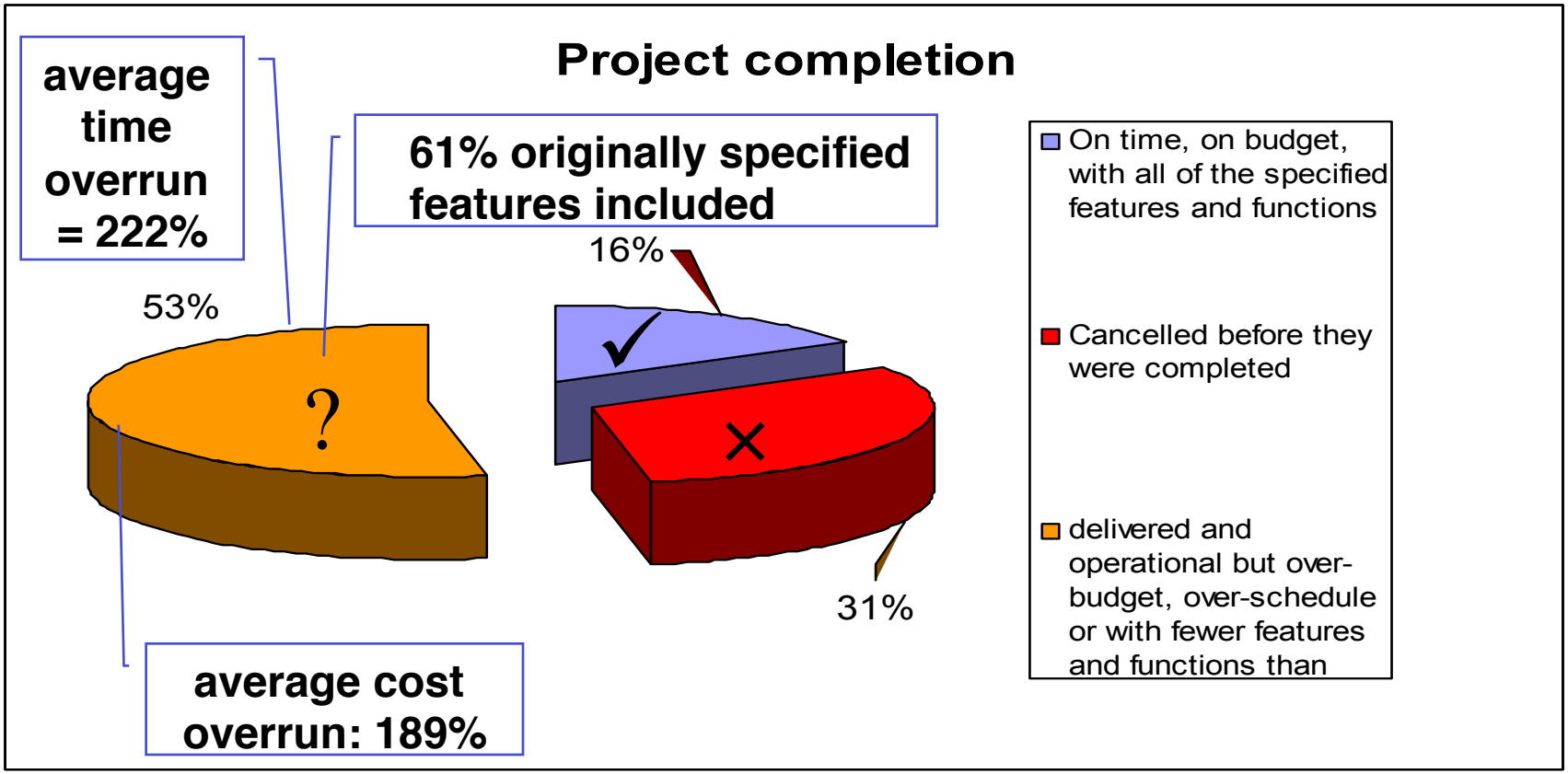
- Listen also to caveats by Sir Anthony Richard Hoare, British Computer Scientist: “Devise clean algorithms!”
- Hoare developed Quicksort algorithm, and formal correctness proofs for programs
- Derived formal language for communicating sequential processes (CSP)
- To define interactions of concurrent processes [3]
- See dining philosophers problem

Chaos Report

- Standish Group – 1995
- 365 IT executives in US diverse industry segments
- 8,380 projects

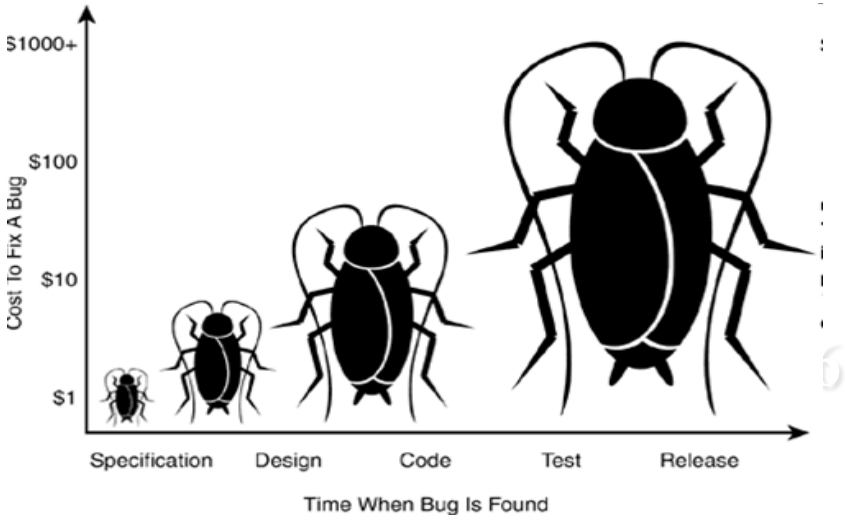
On Average:

- 189% of original budget
- 221% of original schedule
- 61% of original functionality



Symptoms of Software Crisis

- Up to 2000s about US $250 billions spent annually in the US on application development; in 2010s perhaps more
- Of this, about US$140 billions wasted due abandoned projects or reworked SW
- Root cause: not following best SWE practices and standards before 1990s
- Yes, happens today!
- Perhaps less drastically



Symptoms of Software Crisis

- 10% of client/server apps abandoned, or restarted from scratch
- 20% of apps are significantly altered to avoid disaster
- 40% of apps delivered significantly late

Source: 3 year study of 70 large c/s apps 30 European firms. Compuware (12/95)

Observed Problems

Software Products and typical Problems, they:

1. fail to meet user requirements
2. crash frequently; not just *core dump*
3. are expensive, more than projected
4. are difficult to alter, debug, enhance
5. often delivered late
6. use resources non-optimally

Why Such Bad Statistics?

Misconception on software development

- Software myths, e.g., the man-month myth
- False assumptions
- Not distinguishing coding from development of a software product

Software programs have superlinear (AKA geometric) growth complexity and difficulty level vs. size

- Ad hoc approach breaks down when size of software increases

Why Such Bad Statistics?

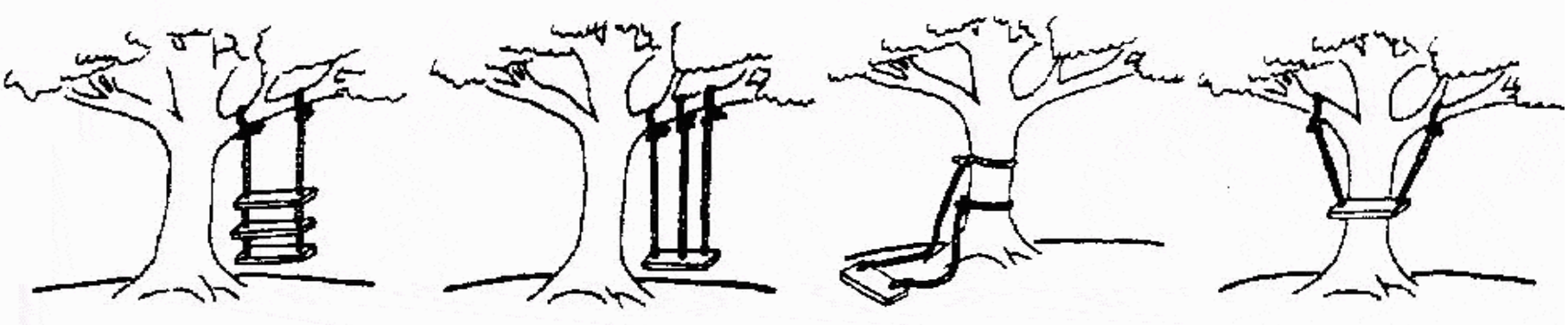
Software professionals often lack engineering training

- **Programmers have skills for programming**
- **But often lack engineering mindset about a process discipline**
- **Personalities can push reason aside!**

Internal complexities

- **Accidents documented by, and some made by: Fred. Brooks**

How SW is Usually Constructed



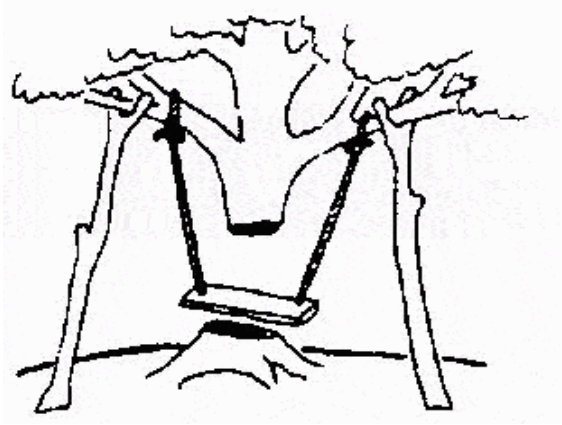
The requirements specification was defined like this

The developers understood it in that way

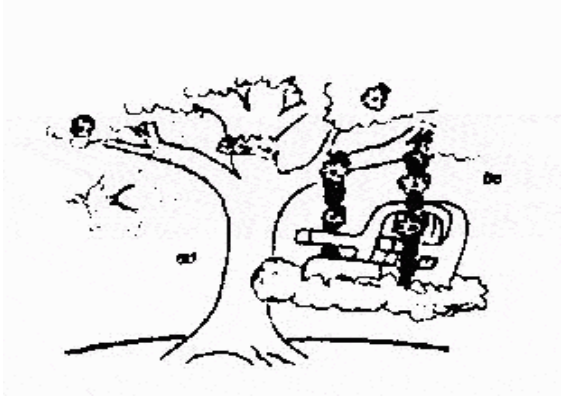
This is how the problem was solved before.

This is how the problem is solved now

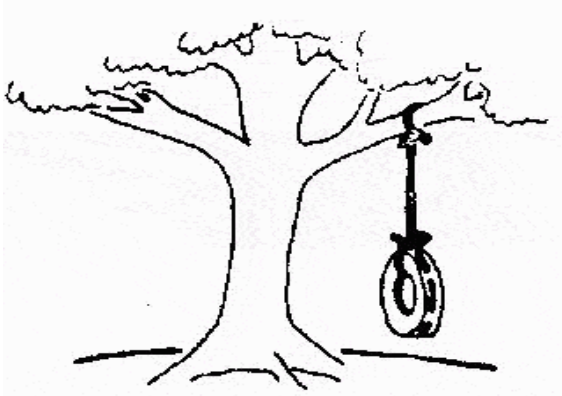
How SW is Usually Constructed



That is the program after debugging



This is how the program is described by marketing dept.



This, in fact, is what the customer wanted … ;-)

Myths: Careful!

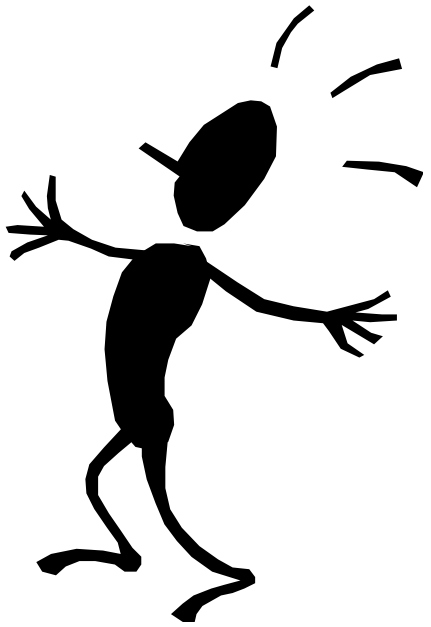
Software Myths

- A general statement of objectives is supposedly sufficient to get started with SW development! (ha!)
- Missing/vague requirements can easily be incorporated/detailed out, as development progresses! (ha!)
- Application requirements can never be stable anyway; software can be and should be made flexible enough to allow changes to be incorporated as they happen! (ha!)

Software Myths

Once the software is demonstrated, the job is done! Yeah!

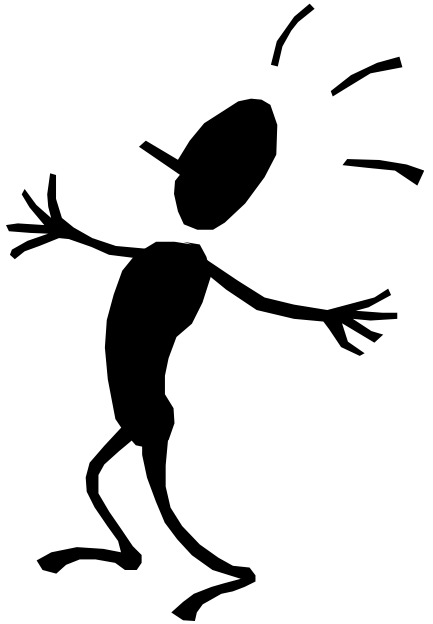
Usually, the problems just begin!



Software Myths

Until the software is coded and is available for testing, there is no way for assessing its quality

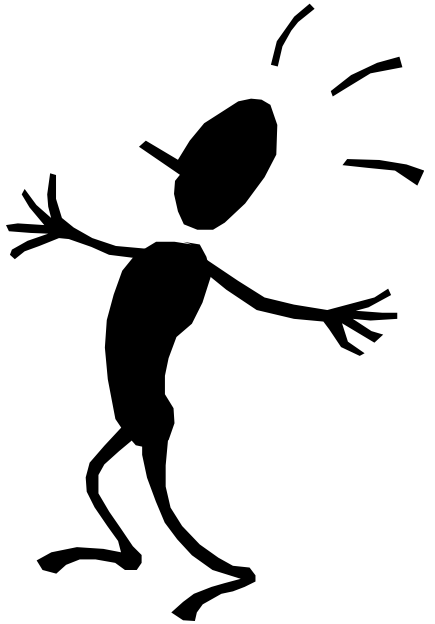
Usually, there are too many tiny bugs inserted at every stage that grow in size and complexity as they progress thru further stages!



Software Myths

The only deliverable for a software development project is the tested code

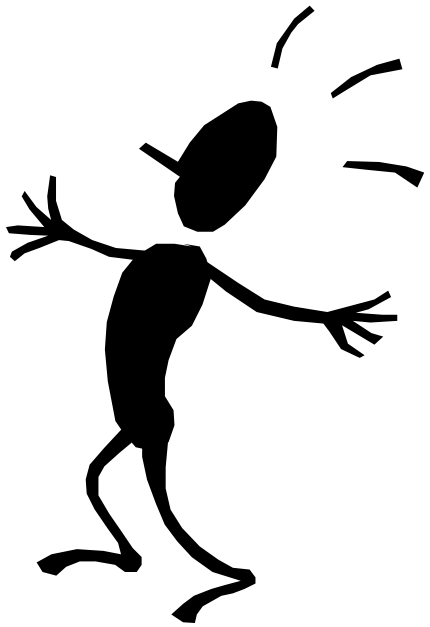
The code is only
the externally visible component
of the entire software complement!



Software Myths

Optimist's approach: “As long as there are good standards and clear procedures in my company, I shouldn’t be too concerned!”

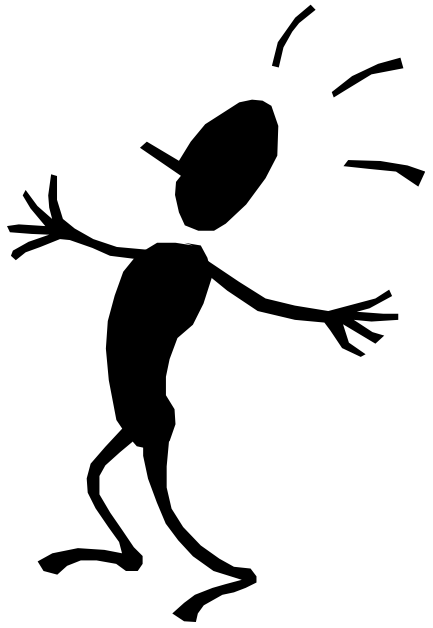
But the proof of the pudding is in the eating; not in the Recipe !



Software Myths

Optimist's approach: "As long as my software engineers(!) have access to the fastest and the most sophisticated computer environments and state-of-the-art software tools, I shouldn't be concerned!"

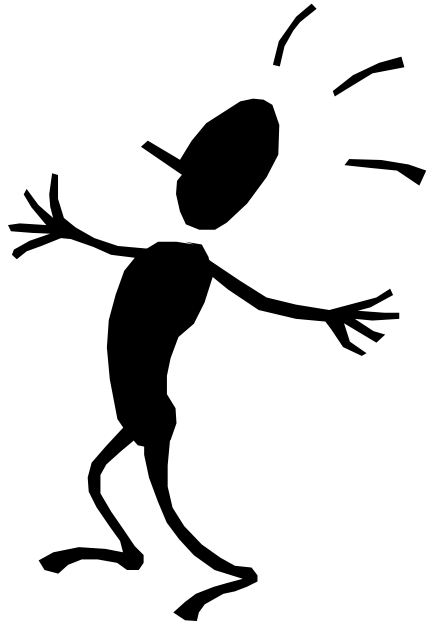
The environment is only one of the several factors that determine the quality of the end software product!



Software Myths

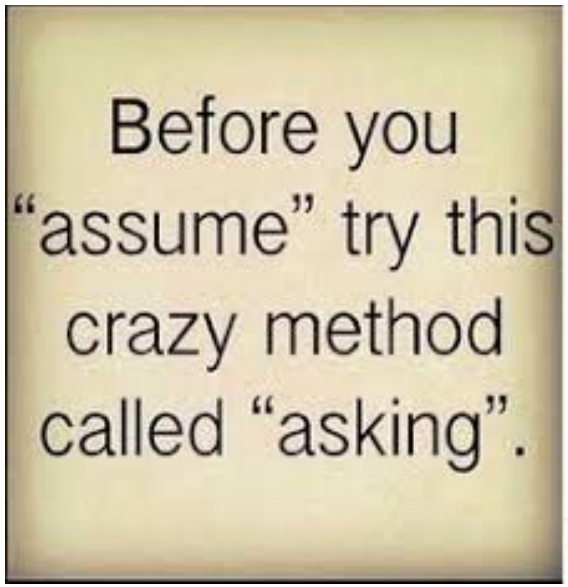
Optimist's approach: “When my schedule slips, I just start a fire-fighting operation: add more software specialists with higher skills and longer experience - they will bring the schedule back on track!”

Unfortunately,
software business does not
entertain schedule compaction
beyond a limit!



Misplaced Assumptions

- All requirements can be pre-specified
- Users are experts at specification of their needs
- Users and developers are both good at visualization
- The project team is capable of unambiguous communication



Ref: Larry

Program – Product Dichotomy

Programs:	Software Products:
Small in size	Large
Author is sole user	Numerous users
Single developer	Team of developers
Lacks clean interface	Clean + clear interface
Lacks documentation	Thorough manuals
Ad hoc development	Managed development

Programming ≠ Software Engineering

Programming: "The process of translating a problem from its physical environment into a language that a computer can understand and obey." *(Webster's New World Dictionary of Computer Terms)*

- **Single developer**
- **"Toy" applications**
- **Short lifespan**
- **Single or few stakeholders**
  - Architect = Developer = Manager = Tester = Customer = User
- **One-of-a-kind systems**
- **Built from scratch**
- **Minimal maintenance**

Programming ≠ Software Engineering

Software engineering: Designing, implementing, documenting, testing mature, robust SW solutions

- Teams of developers with multiple roles
- Complex systems
- Indefinite lifespan
- Numerous stakeholders
  - Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
- System families
- Reuse to amortize costs
- Maintenance: over 60% of overall development cost

What is Software?

Software consists of:

1. **Instructions (computer programs) which, when executed, provide desired function and performance**
2. **Data structures that enable the programs to adequately manipulate information, and**
3. **Documents that describe the operation and use of the programs**

What is Software?

But these are only concrete parts of software that may be seen. There exist invisible parts, equally important:

- **Software is the dynamic behavior of programs on real computers and auxiliary equipment**
- **"… a software product is a model of the real world, and the real world is constantly changing."**
- **Software is a digital form of knowledge. ("Software Engineering," 6ed. Sommerville, Addison-Wesley, 2000)**

Unique Characteristics of Software

1. Software is malleable
2. Software construction is human-intensive
3. Software is s/w intangible and hard to measure
4. Software problems are usually complex
5. Software directly depends upon the hardware

   SW is at the top of the system engineering "food chain"
6. Software doesn't wear out, but it will deteriorate
7. Software solutions require unusual rigor
8. Software has a discontinuous operational nature

   i.e. used during some periods, left unused other times

Casting the Term

1. **Software engineering SWE was born in NATO Conferences, 1968 in response to chronic failures of large software projects to meet schedule and budget constraints**

2. **Since then, the term SWE became popular because software is getting increasingly important to industry and business; yet the actual "software crisis" still persists**

What is Software Engineering?

- **Different focus points exist for this term in various textbooks; here: *IEEE Standard Computer Dictionary*, 610.12, ISBN 1-55937-079-3, 1990**

“The application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software”

What is Software Engineering?

- Software engineering is concerned with theories, methods and tools for developing, managing and evolving software products (I. Sommerville, 6ed.)
- A discipline whose aim is the production of quality software, delivered on time, within budget, and satisfying users' needs (Stephen R. Schach, Software Engineering, 2ed.)
- Multi-person construction of multi-version software (Parnas, 1987)

What is Software Engineering?

- **The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate and maintain them (B.W. Boehm)**

- **The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines (F.L. Bauer)**

What is Software Engineering?

- **The technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost constraints (R. Fairley)**

- **A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers (Ghezzi, Jazayeri, Mandrioli)**

SWE Ethics

Professional Ethical Responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals
- Ethical behavior is more than simply upholding the law

Professional Ethical Responsibility

- **Confidentiality**
  - SW Engineers should respect confidentiality of their employers or clients, regardless of whether or not a formal confidentiality agreement has been signed
  - Upon employment, you usually will sign such a document

- **Competence**
  - SW Engineers should be educated and prepared for their tasks in development environment
  - SW Engineers should not misrepresent their level of competence
  - Should not knowingly accept work outside their competence

Professional Ethical Responsibility

- **Intellectual property rights**
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc.
  - Ensure that all intellectual property of employers, clients, and outside entities is protected

- **Computer misuse**
  - Software engineers should not use tech skills to misuse other people's computers
  - Computer misuse ranges from relatively trivial (game playing on an employer's machine) to extremely serious (dissemination of viruses)

Code of SWE Ethics

- Professional societies in the US have cooperated to produce a code of ethical practice
- Members of these organizations sign up to the code of practice when they join
- Code contains *Eight Principles* related to the behavior of and decisions made by professional software engineers
- Includes practitioners, educators, managers, supervisors and policy makers, as well as trainees and students (you) of the profession

Code of SWE Ethics

Preamble:

- **Short version of code summarizes aspirations at a high level of the abstraction; clauses included in the full version give examples and details of how these aspirations change the way we act as SWE. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, aspirations and details form a cohesive code**
- **Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:**

Code of SWE Ethics

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

Code of SWE Ethics

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

Code of SWE Ethics

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

Conceivable Ethics Dilemmas

- (Possible) Disagreement in principle with the policies of senior management
- (If) Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system
- (Consider) Participation in the development of military weapons systems or nuclear systems

Key Points

- Software engineering is an engineering discipline which is concerned with all aspects of software production
- Software products consist of developed programs and associated documentation
- Essential product attributes are maintainability, dependability, efficiency and usability
- The software process consists of activities which are involved in developing software products
- Basic activities are software: specification, documentation, development, validation, releasing, distributing, and evolution

Key Points

- SW Methods are organized ways of producing software. Include suggestions for processes to be followed, notations to be used, rules governing the system descriptions which are produced; plus design guidelines
- CASE tools are software systems which are designed to support routine activities in SW process such as editing design diagrams, checking source consistency and keeping track of program tests and results
- Software engineers have responsibilities to society and engineering profession. Should not solely be concerned with technical issues

Key Ethics Points

Professional societies publish codes of conduct which set out the standards of behavior expected of members

Summary

- Programming seems easy, yet generating correctly working SW, designing complex SWE solutions to real problems is hard
- CMM is tool to help programmers be disciplined, thus increase reliability of working SWE solutions
- Key SW development steps include: requirements specification, design, review, documentation, testing
- Sir Charles Antony Hoare published, promoted correctness proof of SW
- Clean SW development unexpectedly hard
- Code of SW Ethics published, with guide for SWE to adhere to the stated principles

References

1. History of Linux: https://en.wikipedia.org/wiki/History_of_Linux
2. On Software Crisis: Shttps://en.wikipedia.org/wiki/Software_crisis
3. Concurrent processes: https://en.wikipedia.org/wiki/Concurrency_(computer_science)
4. CMM: https://en.wikipedia.org/wiki/Capability_Maturity_Model
5. On Dijkstra: https://en.wikipedia.org/wiki/Edsger_W._Dijkstra
6. TAOCP: https://en.wikipedia.org/wiki/The_Art_of_Computer_Programming