

# CPT304 – Topic 1

## Software Engineering II



# 1. Software Crisis



# Software Crisis

- The challenges and problems faced in software development, particularly in the 1960s and 1970s, when demand for complex software systems outpaced the ability to design, implement, and maintain them effectively.



# Historical Context

The Beginning of Computing 40s – 50s

- Introduction to the first electronic computers, such as ENIAC and UNIVAC, which were primarily used for scientific and military applications.
- The use of machine or assembly language, which was time-consuming and error-prone



# Historical Context

## The Beginning of Crisis 60s

- The rise of commercial computing and the need for more complex software systems, like payroll and inventory management.
- **Project Failures:** e.g. the IBM OS/360, delays and budget overruns, the growing gap between software demand and development capabilities.
- lack of standardized methodologies and processes.



# Historical Context

The Beginning of Software Engineering 1968-69

- Coining the Term “Software Crisis” in 1968
- The NATO conference acknowledged the inability to produce reliable and efficient software on time and within budget as a significant issue.
- **Call for a New Discipline:** A disciplined approach to software development, laying the groundwork for software engineering.



# Historical Context

## Escalation and Response 70s

- As systems grew more complex, the crisis deepened, projects experiencing significant overruns and failures.
- Introduction of structured programming and the Waterfall model.
- Recognition of the growing burden of software maintenance, which consumed a significant portion of resources and budgets.



# Historical Context

## The Impact of the Crisis

- **Economic Consequences:** The software crisis led to substantial financial losses for companies due to failed projects and inefficient software.
- **Human Factors:** Stress and burnout among developers became common, as they struggled to meet unrealistic deadlines and requirements.
- **Technological Stagnation:** The inability to deliver reliable software hampered technological progress and innovation in various industries.





# Historical Context

## The Path Forward

- **Emergence of Software Engineering:** The crisis catalyzed the development of software engineering as a distinct discipline.
- **Focus on Quality and Process:** Introduction of quality assurance practices, project management techniques, and process improvement models.
- **Legacy and Lessons:** The software crisis highlighted the importance of understanding software complexity, effective communication, and the need for continuous adaptation of methodologies.




# Key Issues

- Project overruns in time and budget
- Software that fails to meet user requirements
- Poor quality and unreliable software
- Maintenance challenges and escalating costs



## 2. The werewolf appears




# No Silver Bullet: Essence and Accidents of Software Engineering

- published in 1986 and has since become a seminal work in understanding the inherent challenges of software development.
- **Author Background:** Frederick P. Brooks, a renowned computer scientist



# Core Thesis

- **No Silver Bullet:** Brooks argues that there is no single breakthrough—no “silver bullet”—that will dramatically improve software development productivity or reliability by an order of magnitude within a decade.
- **Essence vs. Accidents:** He distinguishes between the essential difficulties of software engineering and the accidental difficulties.



*“...become a monster of missed schedules, blown budgets, flawed products ...”, in short “a werewolf” the solution to which is a “silver bullet” that “ ...makes software costs drop as rapidly as computer hardware costs ...”.*

First published as: Brooks, F.P. (1986) “No Silver Bullets”, Proceedings of the IFIP Tenth World Computing Conference, (ed.) H.-J. Kugler, pp 1069-79]



# Essence and Accident

- Essence - Essential difficulties are the inherent challenges associated with the nature of software itself.
- Accidents – Difficulties that are not intrinsic to the nature of software but are instead byproducts of the methods and environments in which software is developed.

# Building Software is hard

- *I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. [Brooks]*



# Essential Difficulties 1

## ■ Complexity

- Vast number of states and interactions within the system.
- An essential property, not accidental
- Difficulties: -
  - Communication
  - Understanding all states, unreliable
  - Extending functions without side-effect
  - Hidden states that constitute security trapdoors

# Essential Difficulties 2

## ■ Conformity

- ☐ Software must conform to human institutions and systems, which are themselves complex and ever-changing.
- ☐ Software must conform because: -
  - ☐ It is the most recent arrival on the scene
  - ☐ Perceived as the most conformable

# Essential Difficulties 3

## ■ Changeability

- Software is subject to continuous change due to evolving requirements, technologies, and environments.
- All successful software get changed: -
  - It is being used at the edge of or beyond the original domain
  - Survives beyond the normal life of the machine it was written for



# Essential Difficulties 4

- Invisibility

- Unlike physical systems, software lacks a physical form, making it difficult to visualize and conceptualize.



# Accidental Difficulties

- Accidental difficulties are the challenges that arise from the current state of technology, tools, and practices used in software development.
- These difficulties are not intrinsic to the nature of software but are instead byproducts of the methods and environments in which software is developed.



# Exercise

- Software complexity leads to many difficulties, what are they?
- In your opinion, why is software perceived as the most conformable element?
- Define essential difficulties and accidental difficulties in the context of software engineering. Provide two examples of each type of difficulty.



# 3. Past Breakthroughs



# A successful past (in 1986)

- High level languages
  - Most important productivity development
  - Reduces accidental complexity
- Time sharing and development interactivity
  - Immediacy allows concentration
- Unified programming environments
  - e.g. Unix, provides a workbench and tools





# Lethal Silver ?

- Better HLL ?
- Object Oriented programming ?
- Artificial intelligence



# Lethal Silver ?

- Expert systems
- “Automatic” programming
- Graphical programming



# Lethal Silver ?

- Program verification
- Environment and tools
- Workstations



## 4. The Promising Attacks



# Build vs. Buy

- Not to build, to buy
- Off-the-shelf for mass market
- Immediate delivery, with less errors
- Low cost, cost distributed among users
- Applicable to all user?
- From software company to consulting company.



# Requirement Refinement & Prototyping

- Hardest part of building a software system is deciding precisely what to build.
- Impossible to specify completely, precisely, and correctly the exact requirements.



# Requirement Refinement & Prototyping

*... one of the most promising of the current technological efforts, and one that **attacks the essence**, not the accidents, of the software problem, is the development of approaches and tools for rapid prototyping of systems as prototyping is part of the iterative specification of requirements.*

- Incremental development--grow, don't build.



# Great Designer

- Great designs come from great designers.
- Best designers produce structures that are faster, smaller, simpler, cleaner, and produced with less effort.
- Great products come from one or a few designing minds, great designers.





# Relevance Today

- **Enduring Insights:** Despite technological advancements since the article's publication, the core challenges identified by Brooks remain relevant.
- **Exercise:** Discuss how contemporary practices like Agile and AI-enhanced tools are used to tackle essential difficulties identified by Fredrick Brooks.



# Essential Reading Material

- No Silver Bullet: Essence and Accidents of Software Engineering, by Frederick P. Brooks