History

- why people did the things they did (research, practice)
- aiming to be able to explain the software crisis

Early Days of Computing

- Before the 1950s, it was mostly analog computers (used e.g. for missile trajectories).
- It's hard to identify the first digital computer.
- It's also hard to identify the first computer.

Digital

- Started around the 40s – weren't fast enough for realtime use, but good for [... missile analysis?]

Pioneering Years

- computers were people, considered clerical work

ENIAC

- heavy, expensive (equivalent of ~$7m)
- used vacuum tubes (no transistors or ICs)
  - 18,000 of them – 1 failed every 10 minutes on average
  - longest continuous period of operation was 116 hours
  - most likely to break when you turn it on

Input with Punch Cards

- 1 card per line of code
- deck of cards was a program, which was then loaded into the machine

Computing Economics

- computers much more expensive than people ($600/hr vs. $2/hr)
  - -> desk checking, peer review, manual execution

Focus on Hardware

- application was hardware and software together – couldn't buy software separately
- term software wasn't used till the 50s – first documented in 1958

Move to Software

- software price increased relative to hardware (up to ~90% of the total cost was software by 1985)

Roots of Software Engineering

- mirrored hardware development
- lots of CS departments emerged from maths departments
- programming was an incidental activity, main focus was
- programmer wasn't a recognised job category until the late 1950s
- 1969 "software unbundling" decision was the start of the software market

SAGE Development Plan

- defence system for US and Canada
- 1,000,000 lines of code (largest before it was 50,000)

ROCKET

- ballistic missile trajectory calculation

IBM System/360 and OS/360

- Over budget
- Over due
- Less capability than promised
- These three things are very common in software development
- 9 pregnant women can't deliver 1 baby in 1 month

The Software Crisis

- Programming initially assumed to be no problem
- Turns out all software was late etc like IBM 360 above

Software Engineering

- early CS research focused on languages and techniques
- Dijkstra “goto considered harmful” in ACM (1968)
- NATO Conference on Software Engineering
  - 1968 in Garmisch(sp?)
  - Can find the report online
  - First SE conference
- “Software engineering” coined as a provocative term – we should be engineering it

Recommended Reading

- The Mythical Man-Month
  - classic
- The Computer Boys Take Over
  - very good, recent
- In the Beginning: Recollections of Software Pioneers
  - Excellent writer
  - Collection of essays from pioneers

Software and Its Development

Types of Software Project

- A solution is a software system

Bespoke Solutions

- developed for a specific customer
  - tailored to their needs
  - e.g. new website for AIB
  - new payroll system for Chrysler
- pull

Market-targeted Solutions

- commercial off-the-shelf
- for a market of customers, not a specific customer in mind
  - e.g. OSs
  - DMBSs
  - Web servers
  - Enterprise resource planning systems (e.g. SAP)
- Normal companies should not be developing these for themselves – waste of resources
- Most open source software is market-targeted/off-the-shelf
- Includes development frameworks though they're not used directly by end-users
  - And also don't run independently – like building a car but purchasing wheels from somewhere else
  - e.g. Node.js
- push

Greenfield vs. Brownfield

- "not quite as black-and-white as it might seem"
- sites that have been built on before vs. sites that haven't been built on before

Greenfield

- completely new project
  - don't have to worry about existing systems and so on

Brownfield

- modification or extension of legacy system

[...]

Embedded Software

- runs on special-purpose devices (e.g. a TV)
- can't separate software from hardware – software usually not portable (doesn't need to be!)
- software development is dependent on hardware development
  - changing hardware design will affect software development

Standalone Software

- Software that runs and depends on standard platforms
  - e.g. Linux, macOS
- Can run independently – function does not rely on other systems
- e.g. Microsoft Office

Distributed Systems

- Systems that run on different nodes
- Physically separated
- Function depends on interaction
  - essential point
- e.g. Web-based systems, air traffic control systems, infrastructural systems like the electric grid

Who Makes All This Software?

Dedicate Software Houses

- companies with software as a core activity
- e.g. DELL/EMC, Microsoft, Apple, Google

Non-Traditional Software Companies

- companies that don't have software as a core business but create software
- you wouldn't think of these companies as typical software companies
  - automotive sector (all car brands)
  - financial sector
  - telecoms
  - home & professional appliances
    - e.g. lawnmowers than mow automatically
- e.g. Rolls-Royce, BMW, Philips
- "Every company is becoming a software company"

Consultancy & Services

- companies that sell their expertise and time to customers that need software-based solutions
  - outsourcing
- companies can be domain-specific
  - e.g. within telecoms, medical, automotive
- e.g. Dell Services (now NTT)
- Kugler Maag (German automotive consultancy)

Individual Contractors

- Independent, self-employed consultants who sell their time and expertise
- Often just blend in with staff developers
  - Sometimes perceived to be "outsiders" by in-house team
- Gives companies flexibility to scale the workforce up/down

Open Source Developers

- Traditionally volunteers
  - but increasingly employed by companies
- Why?
  - Solving their own problems they care about

    - Enjoy hacking
    - Future career prospects (showing off skill)
- More on this later

Key Activities of Software Engineering

- Recognition in 60s that software wasn't an afterthought led to a focus on systematic approaches

Activities

1. Specification
2. Design
3. Code
4. Quality Assurance
5. Delivery & Deployment
6. Maintenance & Evolution

Specification

- define what the system must do
- typical output: software requirements specification (SRS)
- function requirements
    - what should the system do?
- non-functional requirements
    - how should the system do it?

Design

- Create a design for the software that satisfies the requirements
- Typical outputs:
    - high-level or architecture design
    - [...]

Coding

- Implement the software
- Expected outputs:
  - software
  - test suite (potentially)
  - documentation

Quality Assurance

- Test the software

Delivery & Deployment

- Typical outputs:
  - Running system on customer's site
  - Running system on a server
  - Installable software on some media
    * especially market-driven development

Maintenance & Evolution

- adapt and maintain the software
- typically 80% of the cost
- Typical outputs:
  - Updated specification (hopefully!)
  - [...]

A Rational Process for Software Development

- What order should I do things in?

Waterfall Model

- Do things in order, each step interacting with the one before it
- Not a good system (makes no sense)
  - There are better approaches
- When people talk about traditional approaches, they usually mean this (or a variant)
- Also called plan-driven
- Rational logic, not empirical logic
- Very sensitive to changes downstream in the process