**LAS System**
999 calls -> forms -> map ref lookup -> controller (duplicate removal/district multiplexing) -> division controller (vehicle selection) -> activation box -> radio: *3 minutes, 200 staff*

**Problems**
Errors (esp. duplication)
Queues (esp. radio)
Call-backs laborious

**Initial Study**
£1.5m, 19 months
Proviso: packaged, no AVLS
Management forgets provisos

**Bidding**
Firms said proposal unrealistic
Lowest bidder used for £1m, £700k lower than the next..

**Design**
6 month timescale minuted
No formal meeting program or methodology defined
No full time user on project
Systems Options relied on assurances from contractors

**Implement.**
Phase 1 problems: lockup
Phase 2 problems: blackspots in radio, channel overload at shift change, inability to cope with established practices (taking "wrong" ambulance)
Management pressure: go live!
Review asking for volume tests, written implementation strategy, change control and training was ignored
System put in place in one day without any backup or network managers available

**Failure**
Vehicles lost track of
Exceptions scrolled off screen
Incidents held up
Callbacks increased workload
Data delays, voice congestion
Many or none vehicles sent

**"Software Crisis"**
Computer projects had much higher failure risk than others
"Software engineering" coined

**Why Crisis?**
Requirement of perfection
Conform with systems/ standards outside your control
Parts interact in more than 3D
Users demand changes
Hard to visualise software
Software is non-repeating
Requirements change w/ time
Code becomes complex
So high rate of bugs found at start and end of lifecycle

**Big System Problems**
Thin spread of domain knowledge in companies
Communications overhead: hierarchy a good idea?

**Metrics**
Requirements and testing account for 80% of costs
Typically 8KLOC / man year

**Lessons**
Productivity boost comes from using a high level language
Individuals vary 10x difference
Brooks' Law: adding manpower to a late software project makes it later (training + communication costs)

**Waterfall**
Requirements
Specification
Implementation (unit test)
Integration (system test)
Operations / maintenance

**Validation**
Are we building right system?
Feedback path in first half

**Verification**
Are we building it right?
Feedback path in second half

**Advantages**
Management easy (milestone)
Charge for req. changes (even make each stage a contract)
Conducive to good design

**Disadvanta.**
Applicable only where requirements can be defined in detail (e.g. compiler)
Reality isn't like this: iteration is important where tech/law/ requirements/customer environment is changing
Top down quality betterment may be lost over the lifecycle
Safety critical, package software have objections

**Iteration**
Determine objectives/alts
Evaluate alts/resolve risks
Develop and verify prototype
Plan next phases
Fixed # of iterations (guarantees termination)
Increase cost as we spiral
Only use this on relevant parts of system? (e.g. HCI)

**Error**
Deviation from intended state

**Failure** Non-performance of the system w/ some environment

**Fault** Error -> fault -> failure

**Reliability** Probability of failure in period

**Accident** Unplanned event w/ loss

**Hazard** Conditions leading to accident

**Redundancy** Redundant hardware means that software is the failure site
Multi-version programming leads to programs with correlated errors and misunderstandings
Redundant outputs can be confusing to human ops.

**Automation** Computer advises human
Computer interprets output
Computer interprets output and input from human
Human advises computer

**Testing** Cost per bug rises at later stages -> remove them early
Change testers regularly
Due diligence: complying with standards, standard checklist, hire famous consultants…

**HLLs** LOC goes further (10x)
Code easier to read
Appropriate abstraction
Compile time checking
Portability of code
Compilers have errors
Performance may be worse
Manages incidental complexity

**Formal Methods** Forces us to be explicit and check designs in detail
Debate on value for money

**Tools For Management** Activity charts, critical path analysis, PERT, CASE

Chief programmers: focused around a 10x productive guy, but team can only do so much
Egoless programming: code owned by team, not individual
XP: iteration, user interaction
LP: code designed for human

**CMM** Capability Maturity Model
Keep team together
Emphasis shift to process
Repeatable performance
Debug the process

**FMEA** Fault mode/effects analysis
List potential failures and describe the worst case effect

**Fault Tree** Work backwards systematically from identified hazard to primary events to check which are critical or redundant

**Change Control** Compatibility check between versions