**CS2810 Team Project**

**Project Owner** – maintains backlog, priority

**Scrum Team** – Actual technical

**Scrum Master** – Everyone follows the rules + practices of scrum

**Product Backlog Refinement**

* Held before first sprint
* Product owner presents team with vision
* Turn it into user stories
* EVERYTHING could possible put into product
* Owner = gives a business value in each item
* TEAM = amt of effort each item cost
  + how much each sprint = velocity

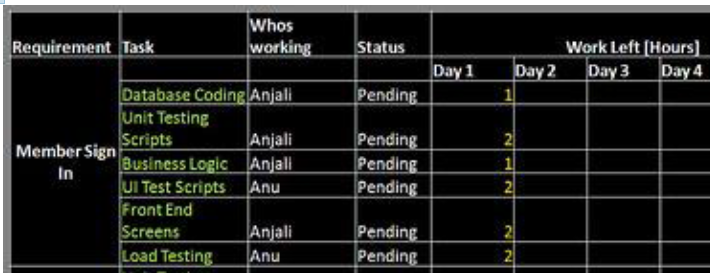
Sprint Planning Meeting

2 parts

Part 1

* Priority setting
* Agreement of item meaning
* Agreement of tests

Part 2

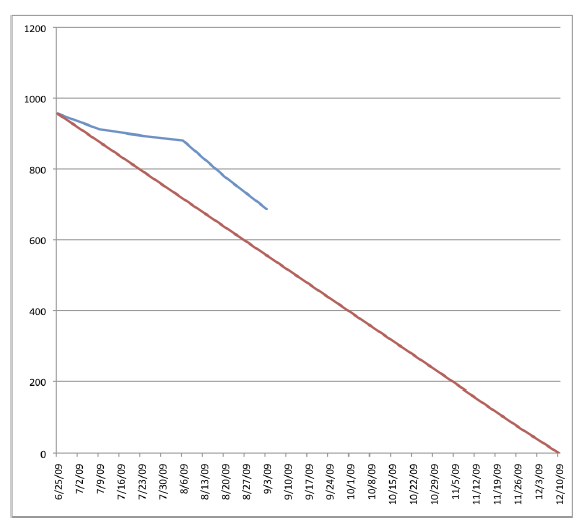
* Time each team mem can commit
* Choose product backlog to complete
* Break product backlog into tasks + estimate effort
* Allocates tasks to team members
* Tracking list:
* 

Scrum Meeting

* What was done
* Planning to do till next meeting
* Problems/impediments
* Short, punctual, nothing else
* Top priority = remove problems

Burndown Chart

* Remaining effort remaining



General Points:

* Communication
* Ideas for features and changes come up
* Fail Fast

Sprint Review

* 5 min presentation to owner
* If no pass of acceptance test = go back to backlog

Sprint Retrospective

* Discussion on what is good/bad

Rules

* Timeboxing
* Focus
* Honesty

Wrong

* Behind schedule
* Cant complete sprint backlog
* End a sprint early

Scrum Management

* Trello
* Board for each sprint
  + Lists = store user stories
  + Card = User story
* Use checklist

Boards

* Product backlog
  + Cards as user stories
* Sprint N

**Architectural Model**

* Software Architecture
  + Link between req and system design
* Architectural design (system)
  + Relevant system components and communication
* In the small (individual programs)
* In the large (complex enterprise)

Advantages

* Communication
* Analysis
* Reuse

4+1 View Model

* Logical
* Process view
* Development
* Physical

Notation

* Informal (annotated in natural language)
* Semiformal (UML)
* Formal (precise semantics, formal analysis)

Characteristics

* Performance
* Security
* Safety
* Availability
* Maintainability

Pipe and filter architecture



Client-Server architecture



Peer to Peer Architecture

Layered Architecture



Service Orientation Principles

* Reusability
* Composability (can form composite services)
* Loose coupling
* Contract (comms agreement)
* Abstraction (hide logic)
* Autonomy (control over hidden logic)
* Discoverability (designed descriptively)

Application Architectures

* Starting point if architectural design
* Design checklist
* Organise work
* Assessing components for reuse
* Vocabulary for communication

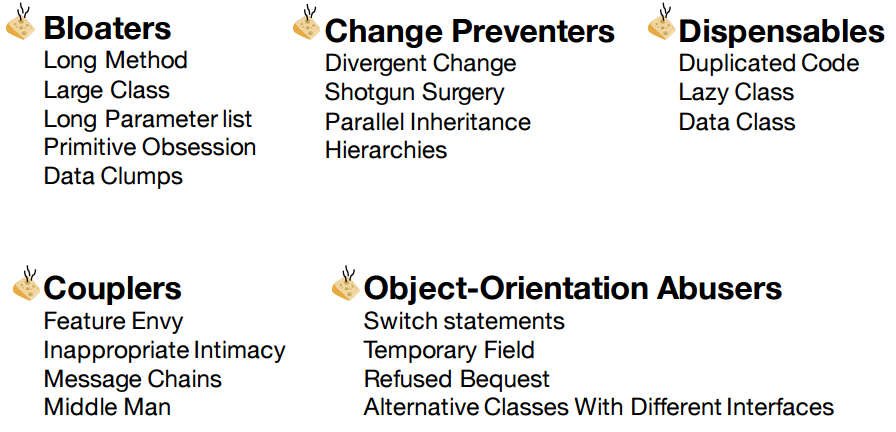
Architectural Models and Agile Dev

* Not necessary to create all
* Not to avoid design

Refactoring

* Design pattern = for change
* Improve Code
  + Prevent “Decay” from changing req
  + Clean up and simplify code, increase readability
  + Improve extensibility

Code Smell

* Indication for refactoring
* Bad practices
  + Unreadable code, complex
  + Hard to uds for others to work with
  + Hard to maintain
* Not Bug, but warning Sign

**WorkFlow**

Simplify & test

• Determine how to refactor the code

• Apply one step of refactoring

• Run tests to ensure things still work correctly

Correctness

* Test Well
* Review Changes (small changes = break codes)

Risk of Refactoring

* Dangerous
  + Introduce new bugs
* Expensive
  + Time spent in dev without external visibility
  + Retesting required
* Easy to waste time
  + Avoid being sidetracked
  + Focus on 20% that benefit 80%

Refactor when

* Need to perform changes
* Add a method/class
* Fix a bug
* Code smells

Don’t Refractor when

* Close to deadline
* API been published
* Someone working on code
* No tests, or tests are failing
* Code doesn’t need to be changed (legacy)

Strategies

* Prepared
* Refactor effectively

**Professional Issues**

Computer Scientist =

* Technical competence
* Judgement
* Professionalism

Ethical Issues FORCC

* Focus
  + Definite goals (don’t change)
  + Next step (don’t worry)
  + Use whatever (meet goals)
* Openness
  + Git/trello
  + Cannot Do = tell your team
* Respect
  + Some may need help
  + Someone cannot complete task
  + Scrum master = not boss
* Courage
  + Choose task, and accomplish task
  + Experiment and try
  + Git will keep old version
  + Fail fast
* Commitment
  + Commited, no boss

Dependability

Reliability = link to dependability

Attributes

* Availability (Readiness)
* Reliability (continuation of correct service)
* Safety
* Integrity
* Maintainability

Threats

* Fault
* Error
* Failure

Means

* Fault tolerance
* Fault prevention
* Fault removal
* Fault forecasting

Fault -> Error -> failure (not necees)

80-20 rule

Software Quality = defects

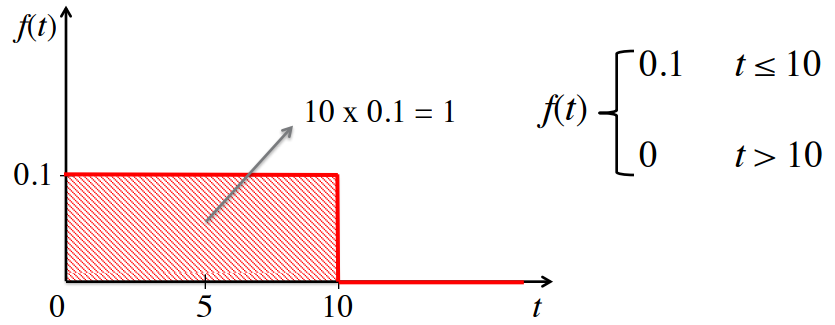
* No. of defects/size
* Time to fix each

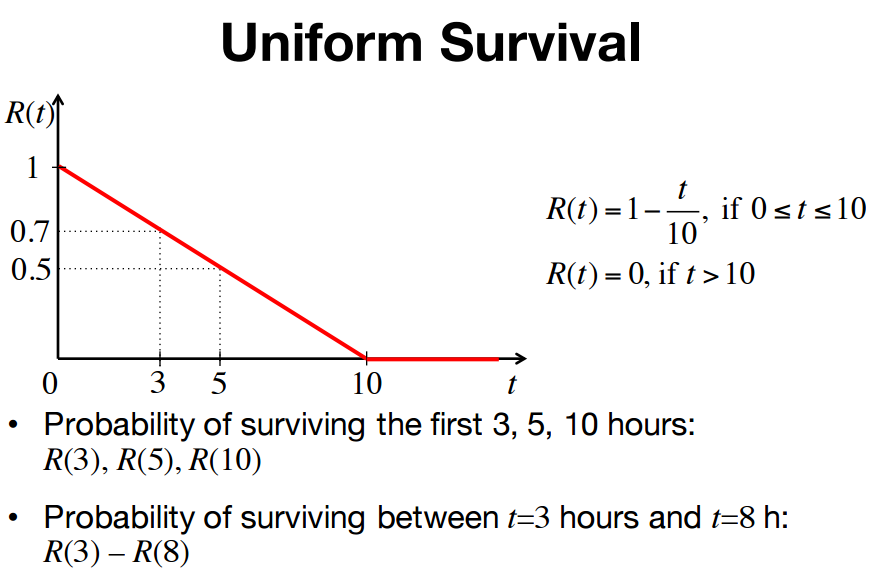
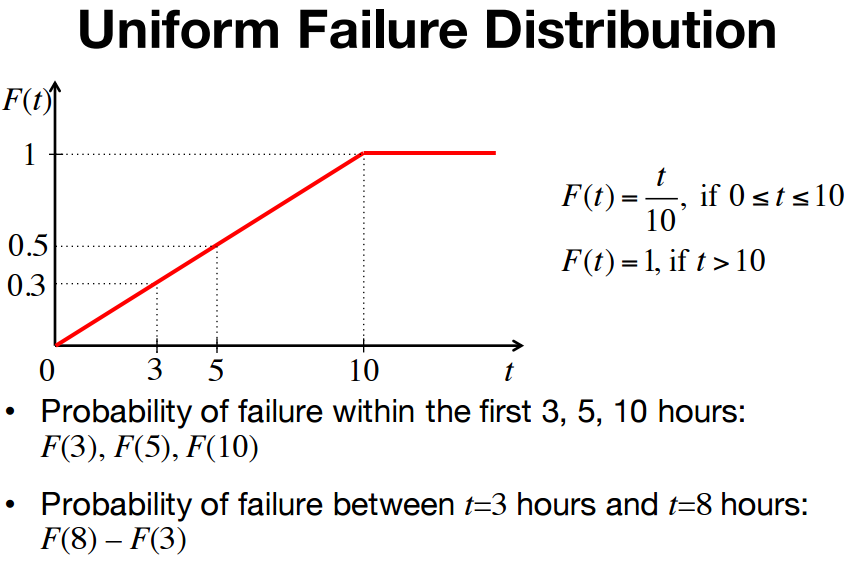
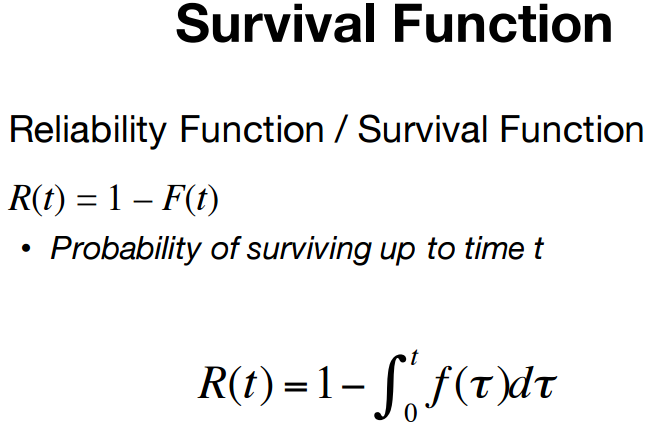
Software reliability = quantitative study of operational reliability of software

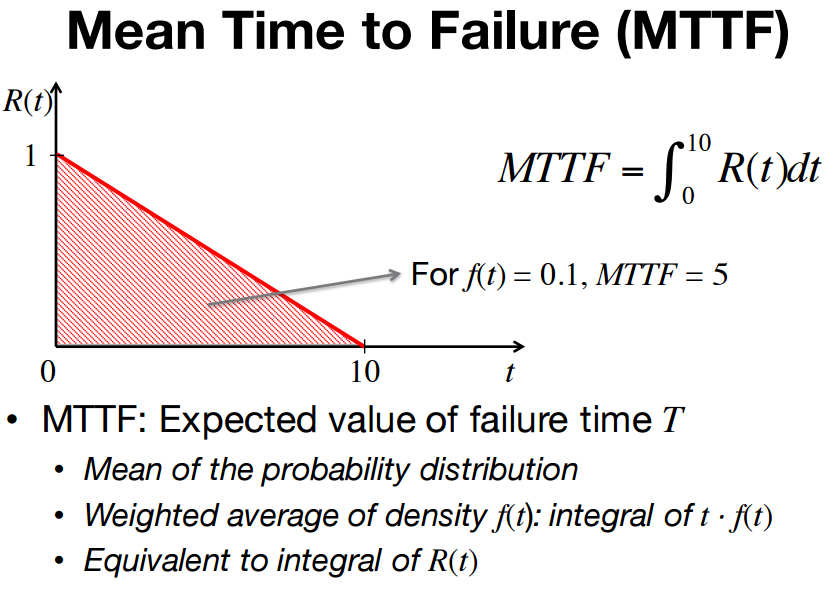
Predicting Failure time

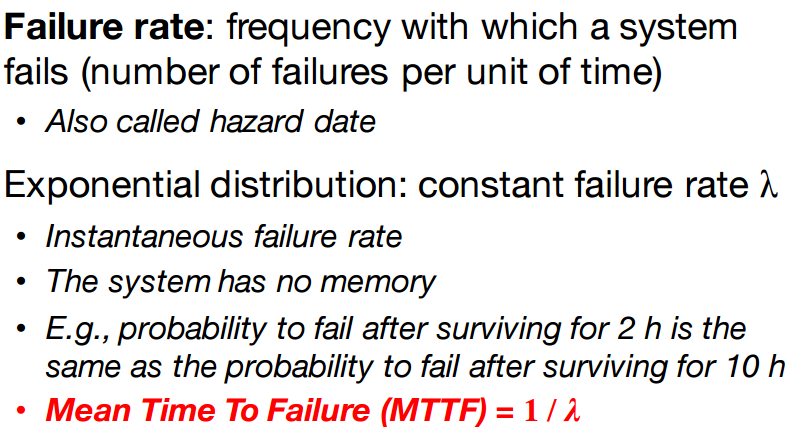
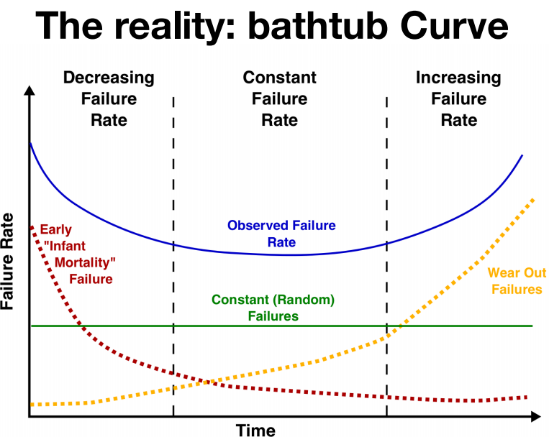
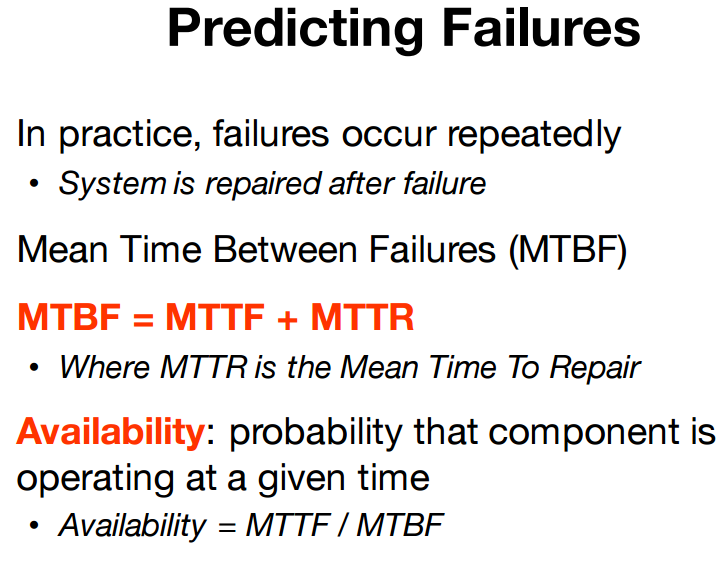
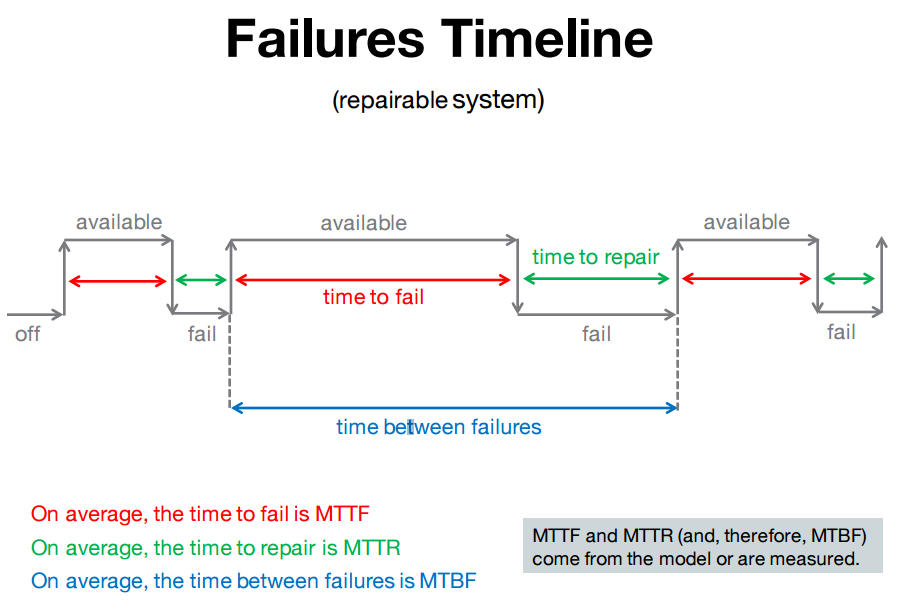
* Failure occurrence are probabilistic in nature
* MTTF

Probability Density Function

* Graph of failure over time
* 
* 

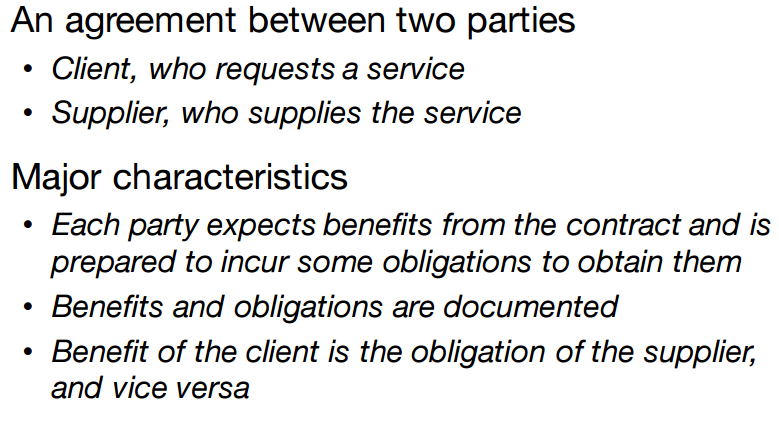
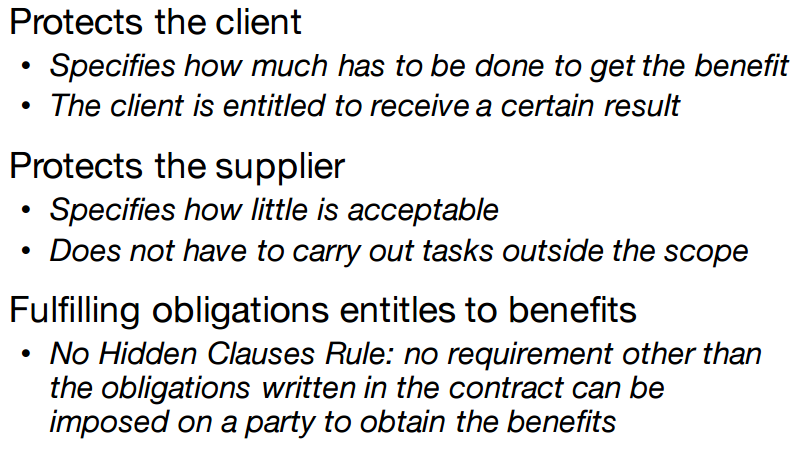
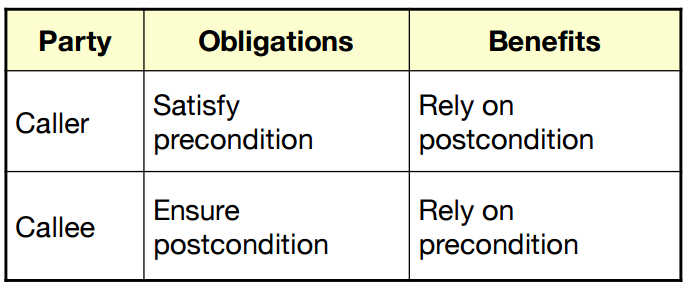
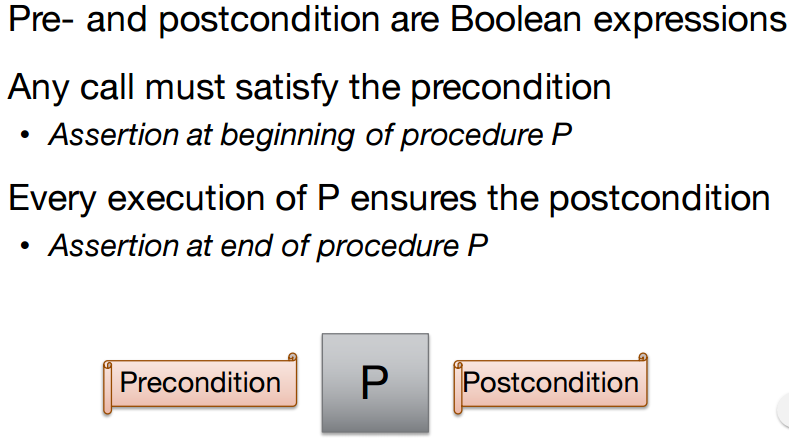
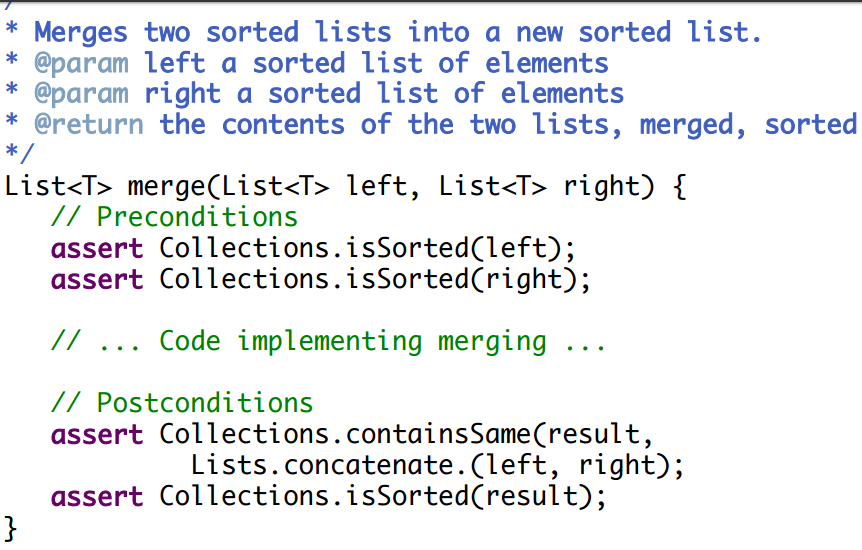
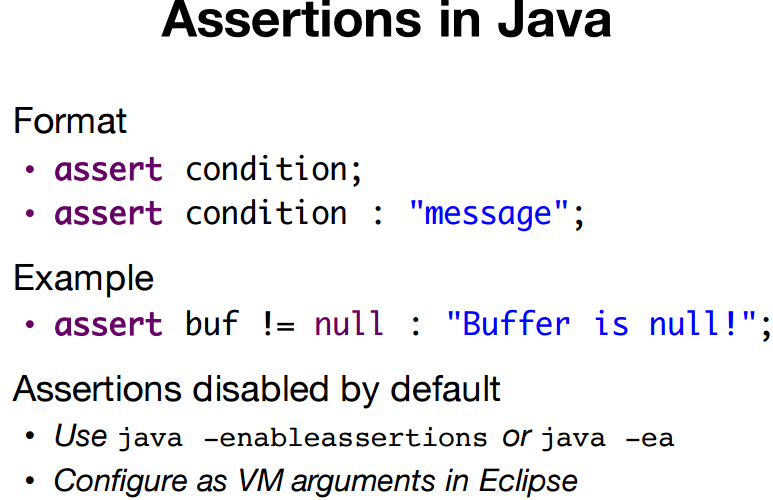


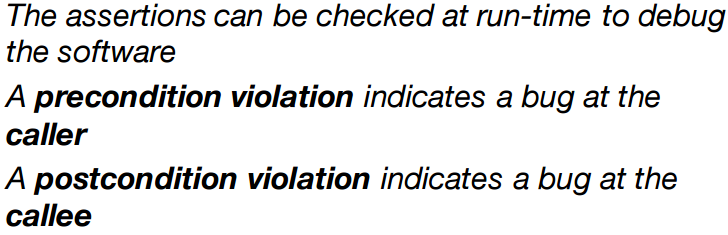
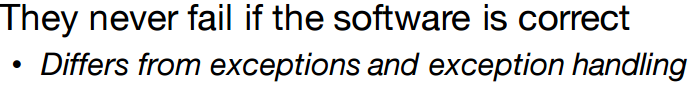
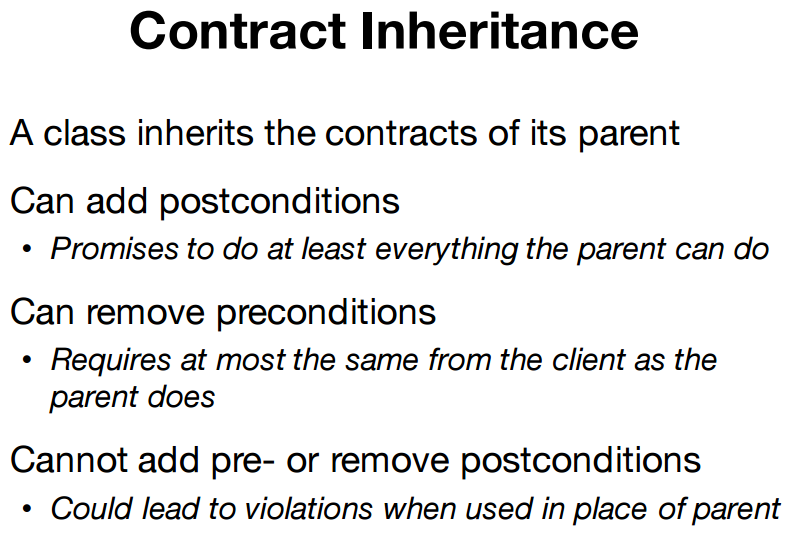
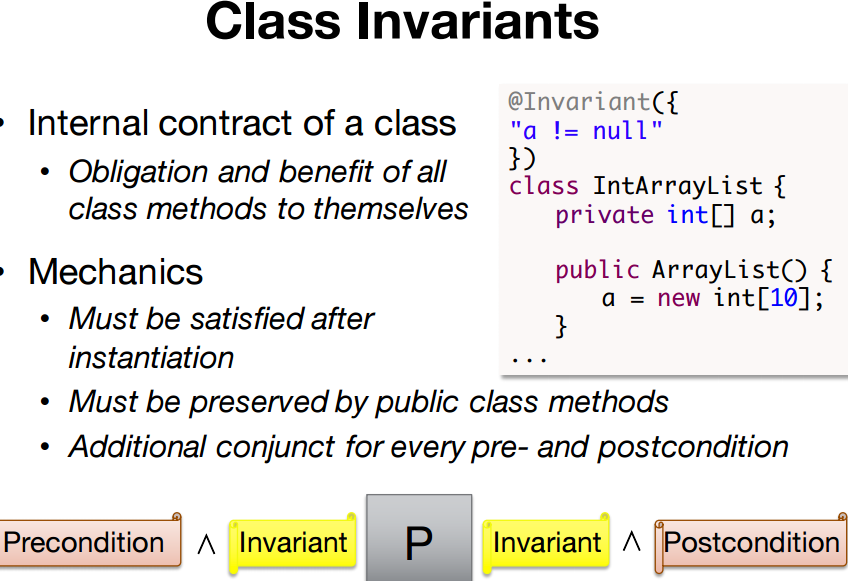
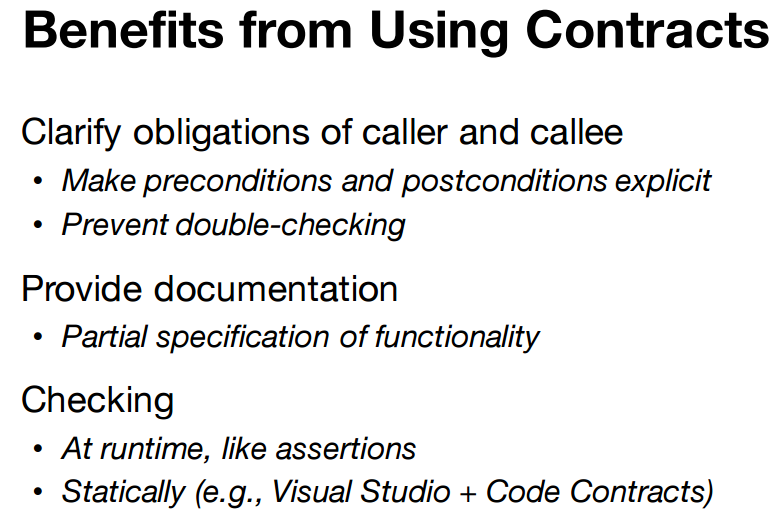
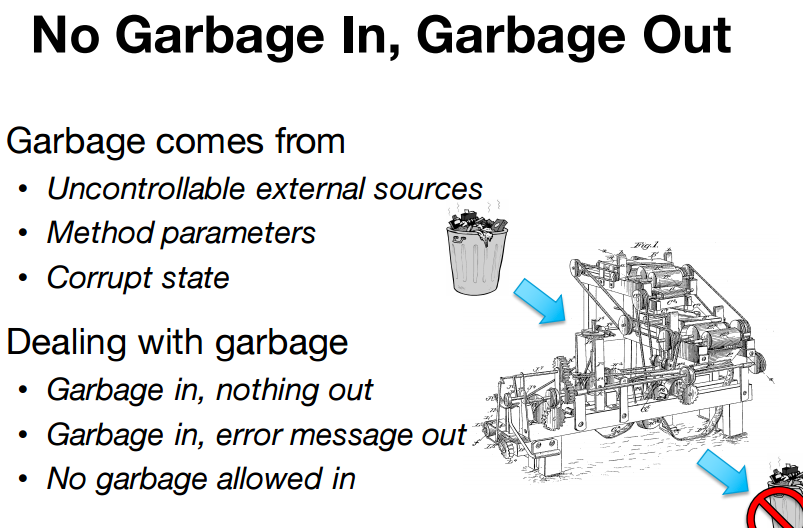
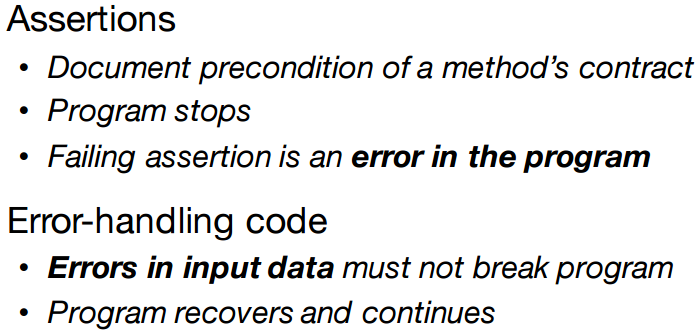
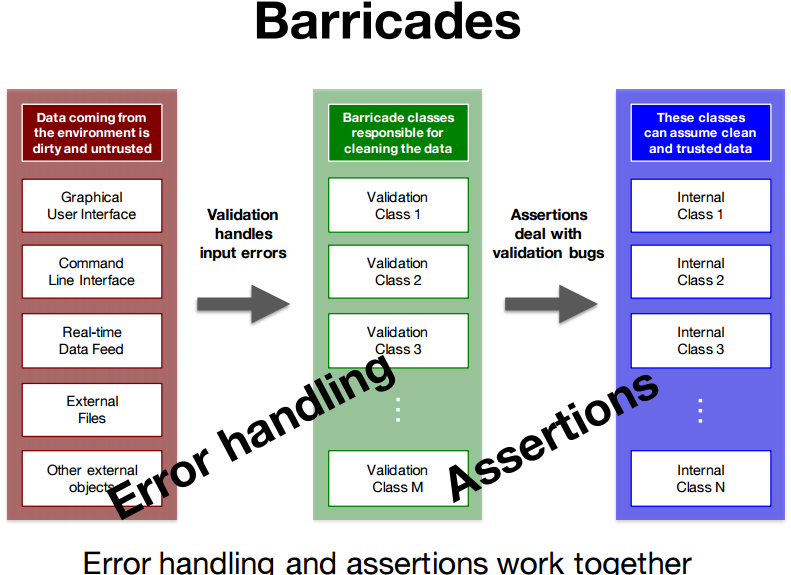
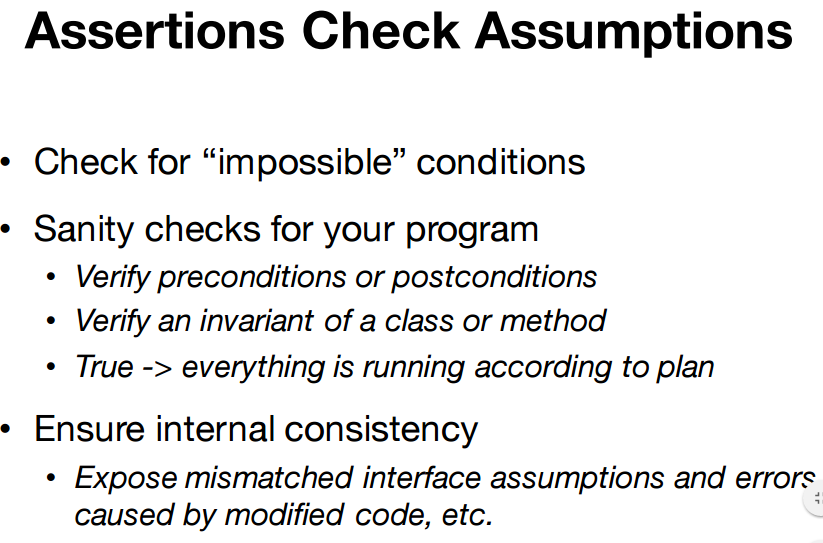


Robust Code

* Stable, no random crash
* Handles corner cases
* Reacts gracefully to unforeseen events/inputs

**Contract**     

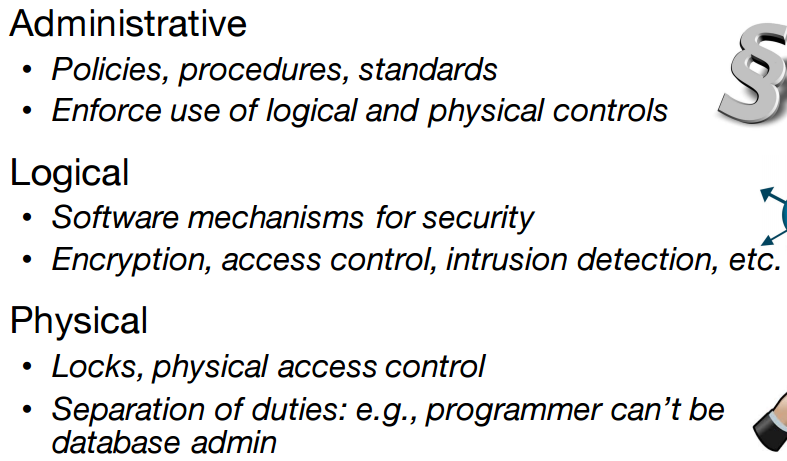
**Violations**         

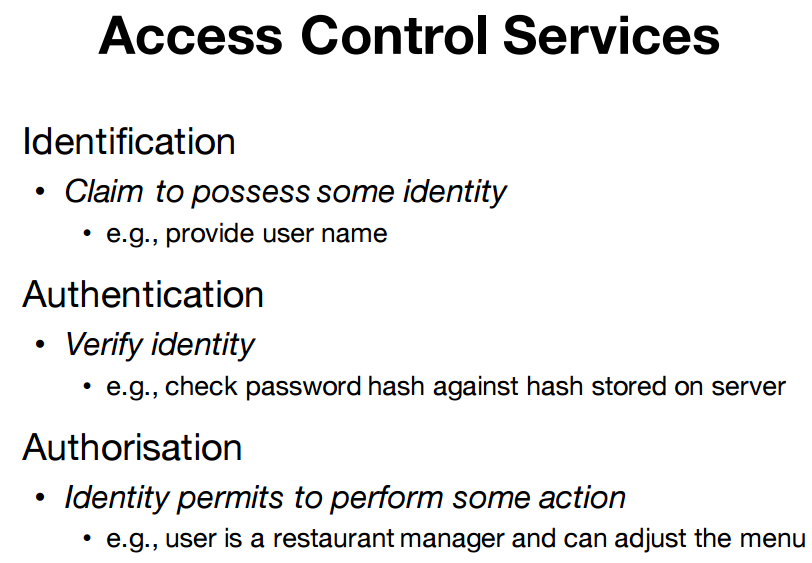
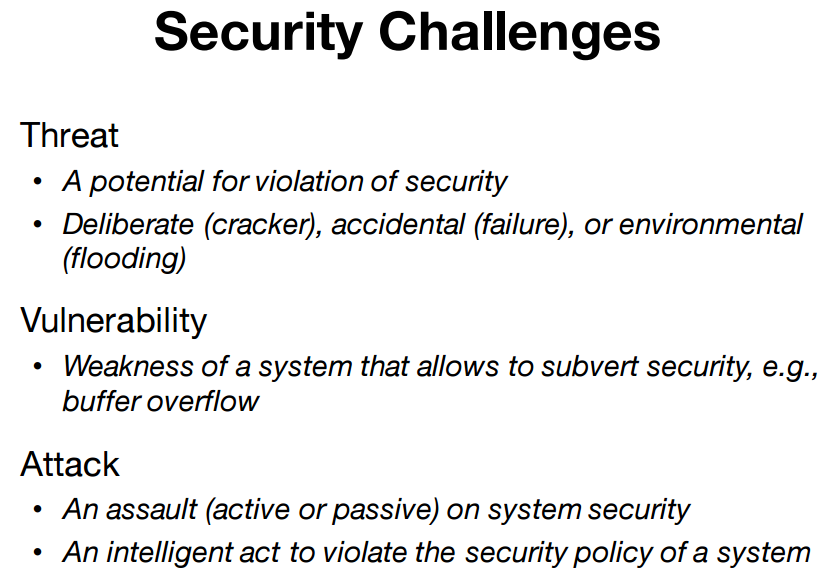
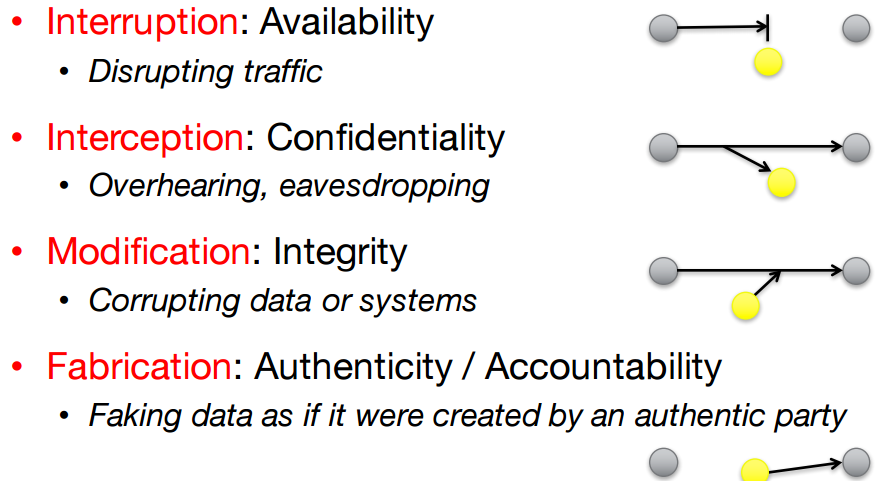
<missing stuff here>

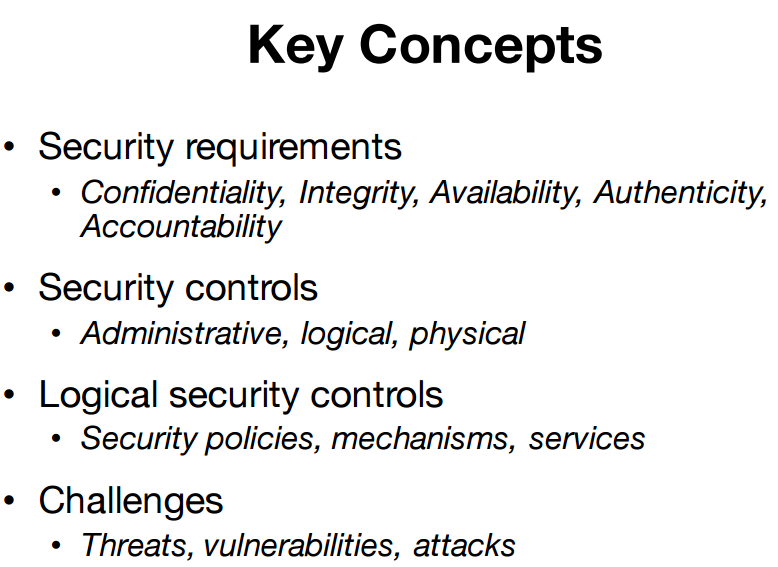
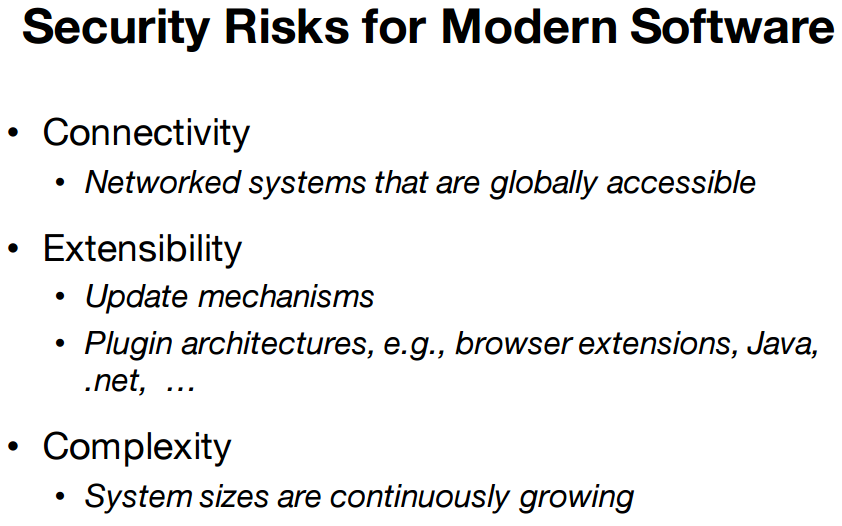
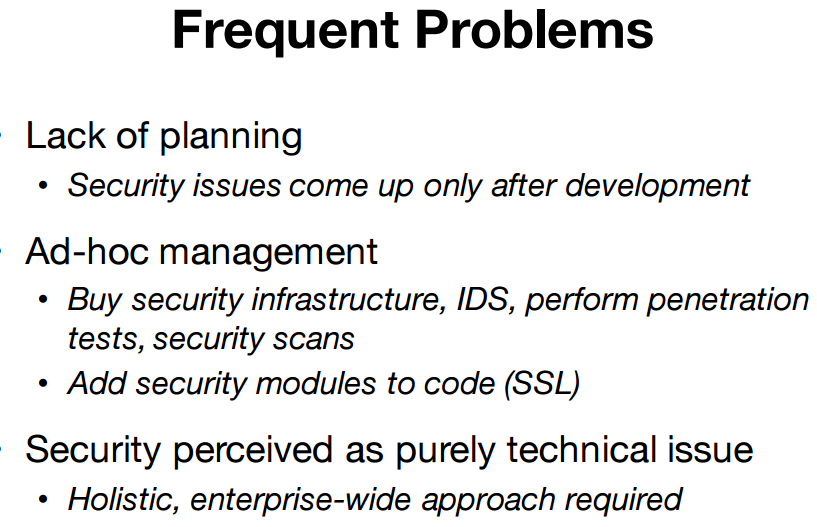
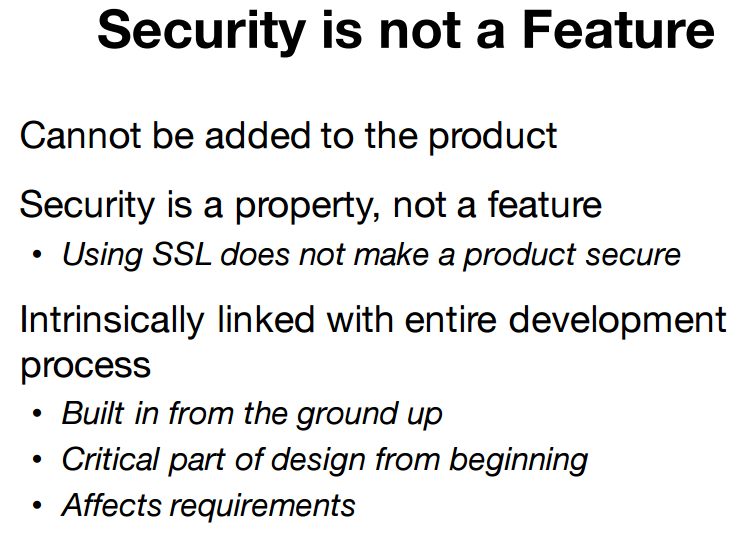
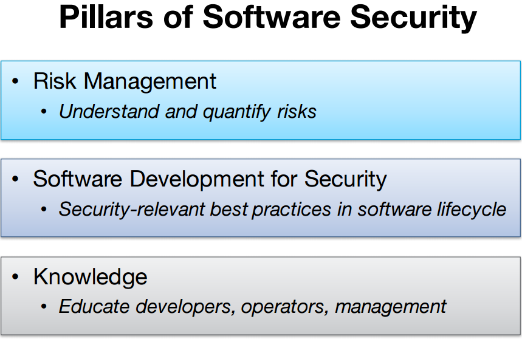
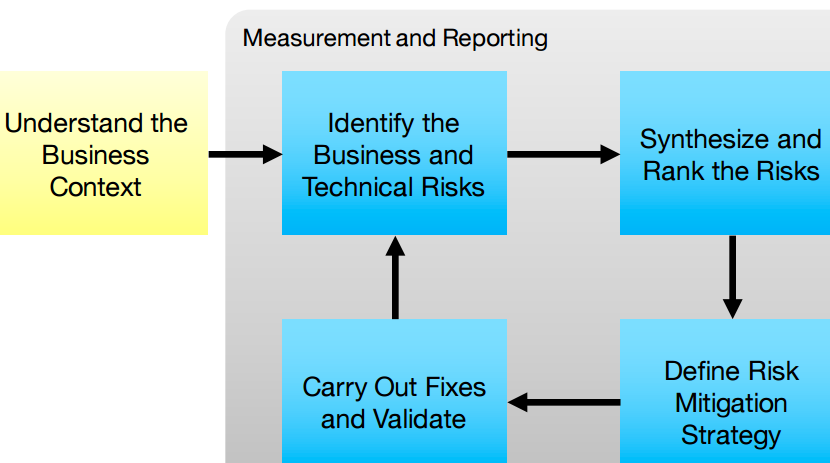
**Software Security**

CIA (Authenticity, Accountability)

Controls



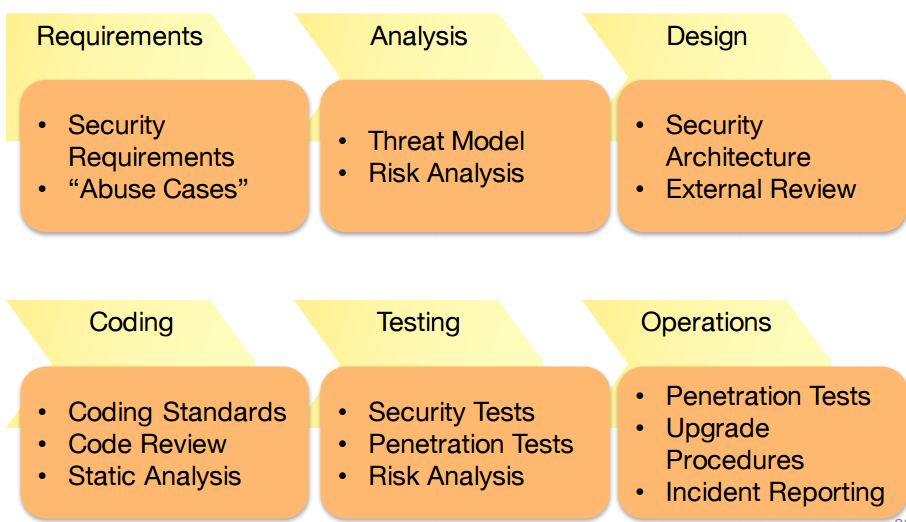
     

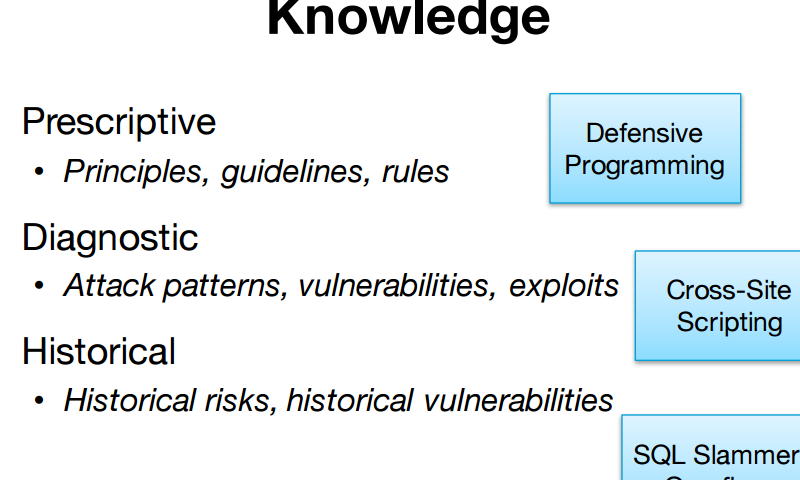
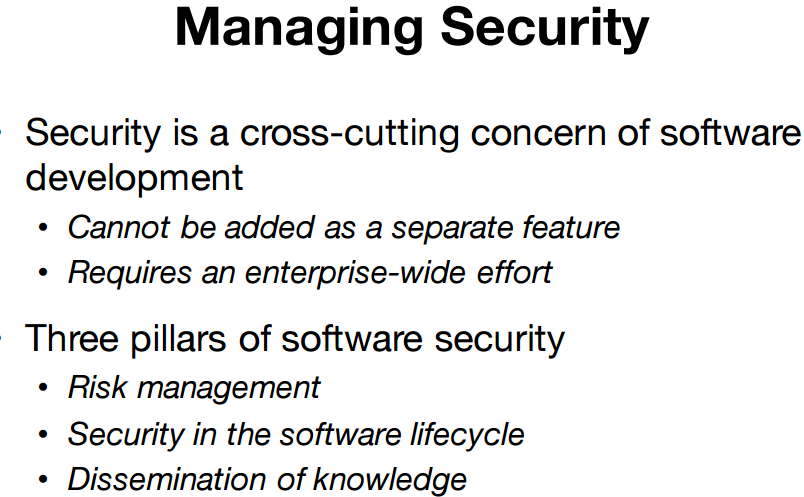
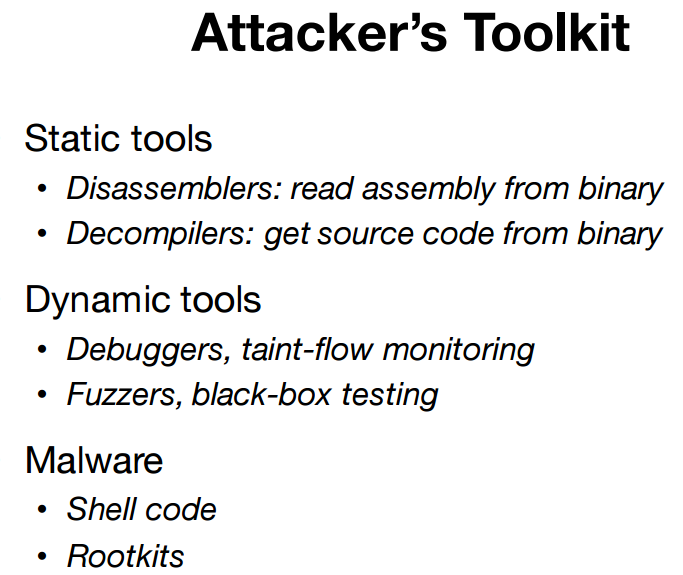
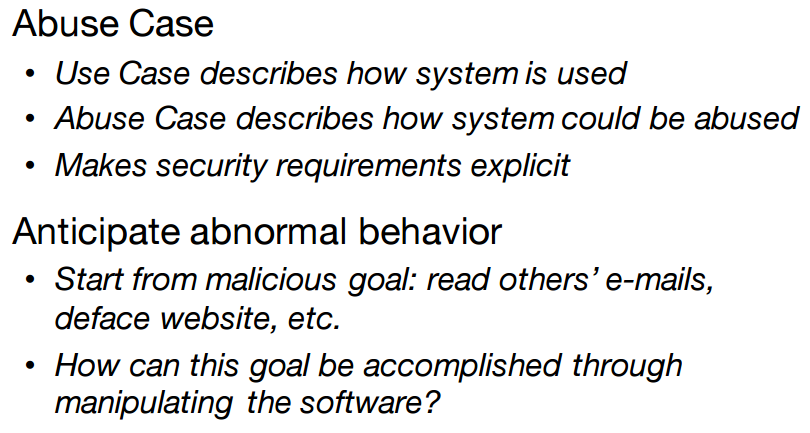
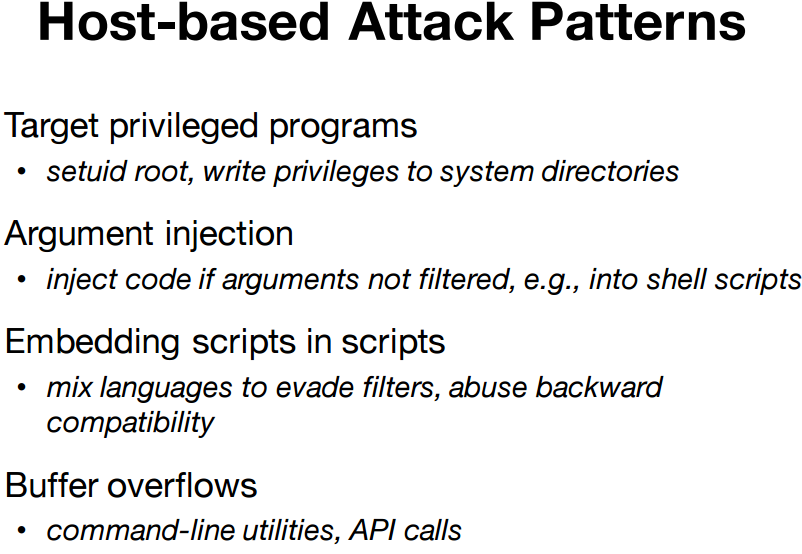
Rank Risks

* Likelihood
* Severity
* Impact
* No. of risks

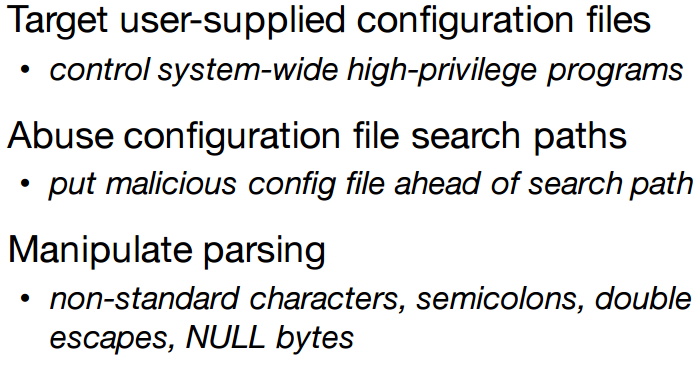
Mitigation

* Costs, likelihood of success, completeness, impact (damage avoided)
* Validation
* Keep repeating (new threats)

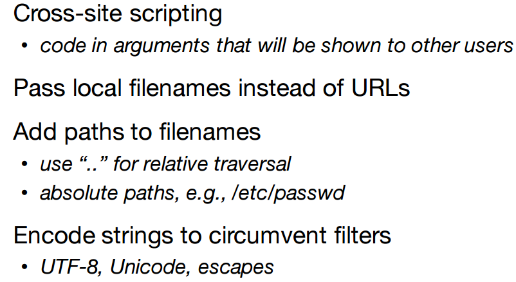


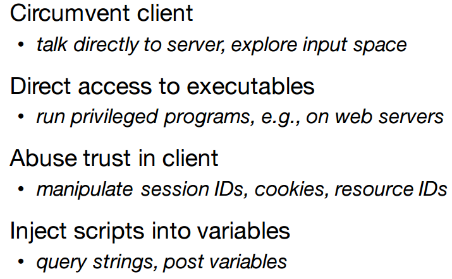
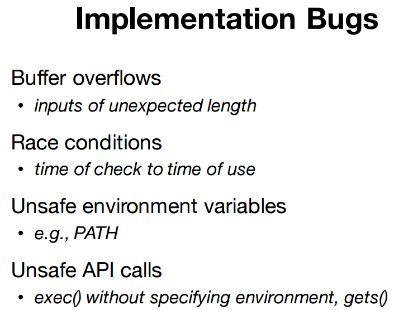
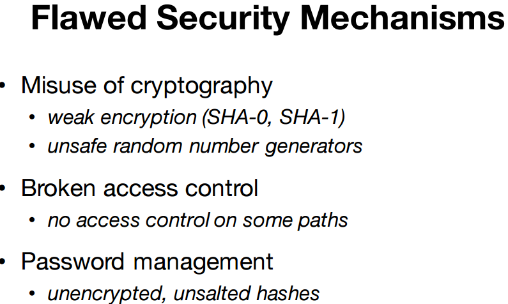
    

**Config attacks**



**Input attacks**



**Client server attack**   

**Good software engineering**

* Code quality
* Encapsulation
* Defensive programming
* Least privilege
* Privilege separation
* Defense in depth
* Secure design
  + Weakest link
  + Fail securely

Measure software

* Efficiency
* Readability
* Maintainability

Diff solutions = diff program

Approach = 1 attr at 1 timeInterna

Software size metrics

* Length (line of Code, LOC, NCLOC, Exec Statement, Delivered Source)
  + Limitations: **code & lang dependent** & **reuse check**
* Functionality
  + **Code neutral**
  + Derived from specs (understood by everyone)
  + Elementary process = smallest unit of activity = meaningful to end user
    - Types (data in motion/rest)
  + Unadjusted function point count (UFC
    - Internal logical file (master files, DB)
    - External interface file
    - External input
    - External output
    - External inquires
* Complexity
  + Program, algo, structural, cognitive
  + Weight = complexity X category
  + UFC for technical complexity
    - Complexity factors

**FP = UFC \* TCF**

* Can use to estimate 2 person-day per FP

No 1 formula

International function point user group (IFPUG)

* Maintains counting practices manual

**Data Element Type (DET)**

**File Type Reference (FTR)**

**Record Element Type (RET)**

Counting EIs:  
For each EI:

* no . of DETs
* no. of types of files

EO: intent to present info to user through logic

Counting EOs:

* Count involved DETs and FTRs

EQ: present info to user through retrieval of data

ILF: user-identifiable info

Counting ILFs: Count the number of DETs  
(sub groups) called Record Element Types

EIF: maintained outside the boundary of app

Functional Vs non-Func Req

FPA analyse costs, time, value of tools, check KPI

Other approaches: Use cases, User Story

**Bug**

Bug tracking tool

Approach to report bug

* clear, understandable
* problem report
* support evidence, concise
* respectful tone
* one bug per report
* if bug in db alr

Bug report fields

* reporter
* assignee
* cc
* ver
* component
* platform
* OS
* Severity (blocker, critical, enhancement)
* Summary
* Steps to reproduce
* Freq (likely, unlikely)
* Priority (P1 – P5)
* Lifecycle (unconfirmed, new, assigned, resolved, verified, closed)
* Resolution (fixed, invalid, duplicate, wontfix,worksforme)

Bugzilla