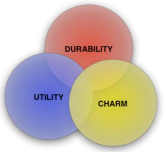
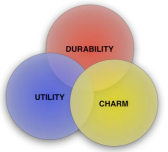


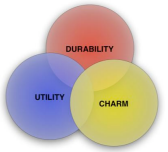
Class 10 CIS 573
(lecture 9)

Gregg Vesonder
University of Pennsylvania
Penn Engineering - Computer & Information Science



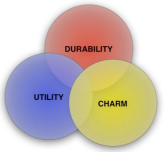
Roadmap

- Finish Brooks
- Real Time Software Engineering
- Reliability
- SOA
- Security
- Review
- Readings this class: Sommerville chapters 15, 30 & 31; Brooks Chapter 16-end and Andersson, et.al., chapter 14
- Readings next week –NONE!!!



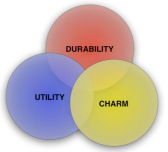
Critical Dates

- Every class project review
- ~~July 23rd Mid Term~~
- August 6th log books due – by Midnight
- August 11th project presentations
- August 13th Final



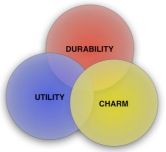
Teams

- Team 1 – Klein Keane, Beck, Buchman, Richardson, Nunez
- Team 2- Wilmarth, Caputo, Xiang, Francis, Nanda
- Team 3- Noronha, Fang, Huang
- Team 4-Whitehead, Liu, Ratnakar



Project Reports

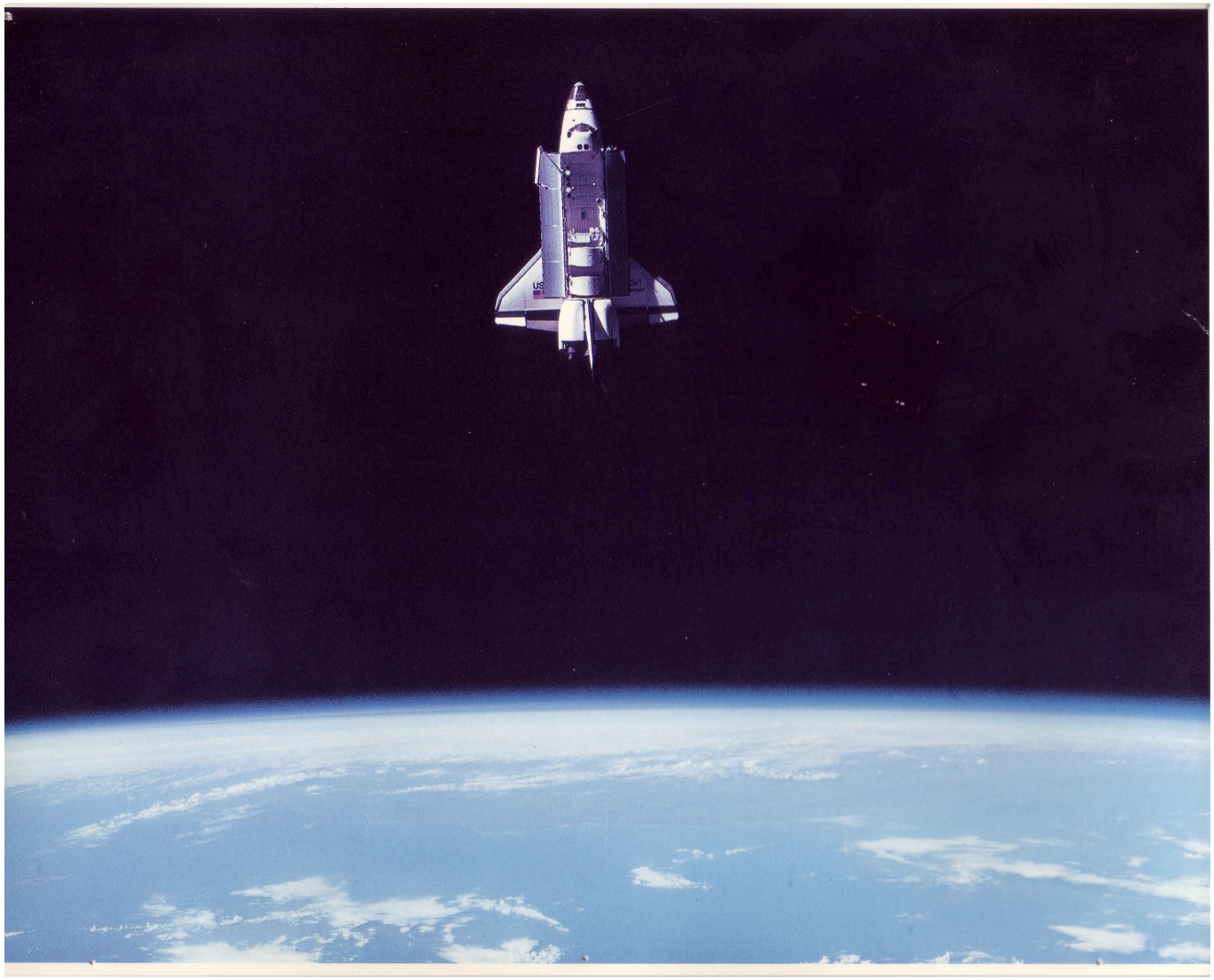
- Presentation each class
  - Green, yellow, red –simplified model + gaps
  - Current pressing issues
  - What was done since last class
  - What will be done before next class
  - Gaps



Clarification: Software Archeology

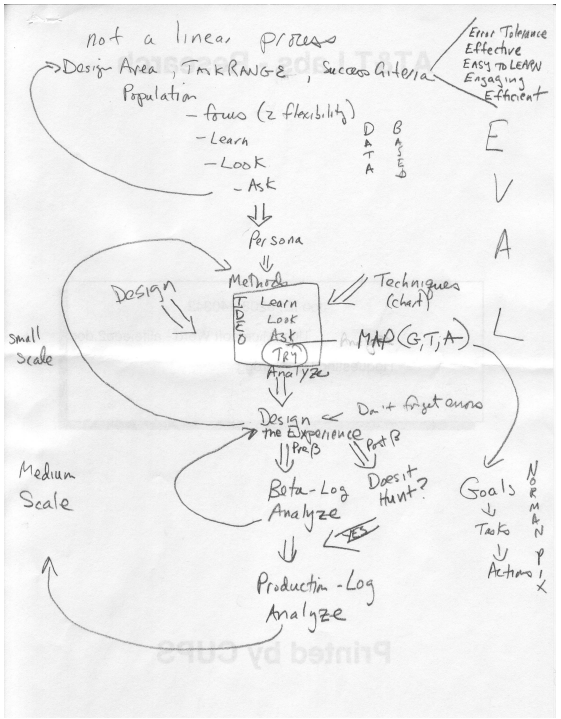
- Trying to understand what was in the minds of software developers using only artifacts left behind
- Hampered by the fact that the artifacts were not created to communicate to the future
- Only part of what was originally created has been preserved
- Relics from different eras (versions) are intermingled
- Dynamic and static techniques
- Support for refactoring
- Browse the workshop! OOPSLA 2001 http://www.visibleworkings.com/archeology/

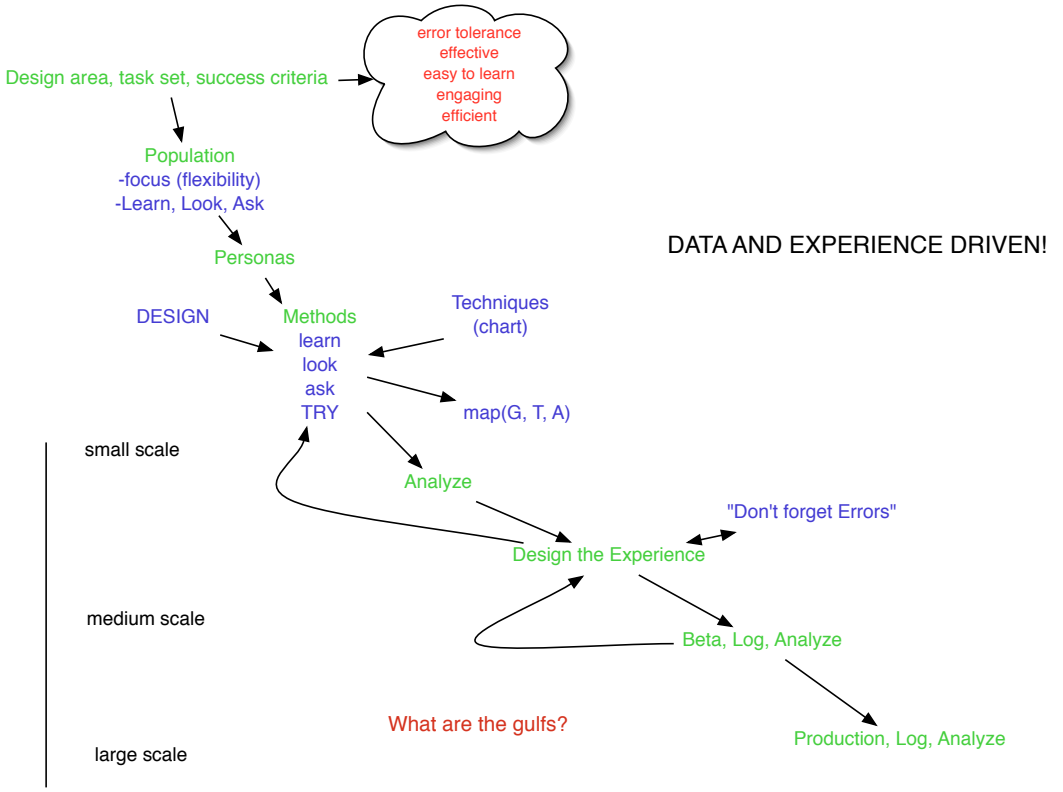
Log Book – Challenger



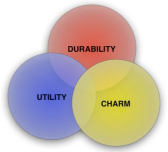
In Summary

- User Experience Design is neither linear nor rigid!





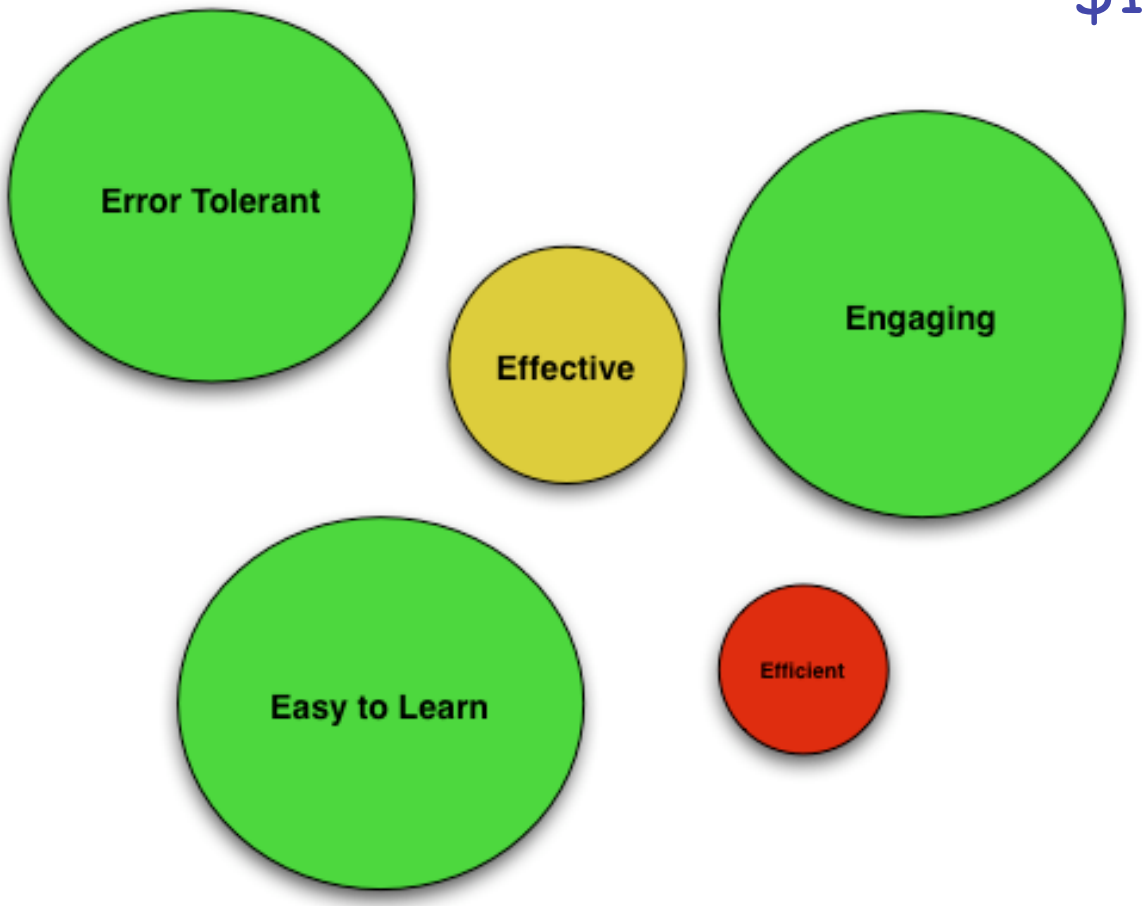
Evaluate User Experience, 5 E's

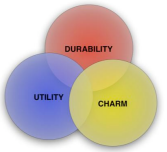


DIMENSION	KEY NEEDS	DESIGN TACTICS
Effective	Accuracy	Focus on places in the interface for potential error and protect against them. Look for opportunities to provide feedback and confirmations
Efficient	Operational Speed	Present only most important information. Work on smooth, direct navigation. Interaction style should minimize actions required
Engaging	Attract users	Consider what aspects of the product are most attractive and incorporate into design
Easy to learn	Just-in-time instruction	Step by step interfaces that help users navigate through complex tasks. Provide training in small chunks if possible
Error tolerant	Validation	Look for places where selection and calculators can replace data entry. Error messages provide opportunities to correct problems

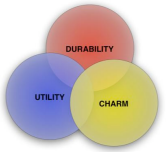
Success Criteria, 5E's
(rational weighting)

$100



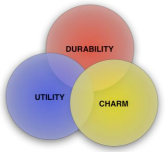


BROOKS



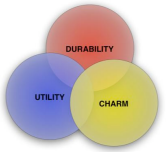
Brooks
No Silver Bullet (1986)

- Accidental vs. essential (abstract conceptual structures). Suggests:
  - Exploit mass market - buy vs. build, recurrent theme
  - Use rapid prototyping for establishing software requirements
  - Grow software organically (incrementally)
  - Identify and develop the great conceptual designers
- Software projects as werewolves - turning from the familiar to the horrible
- There is no silver bullet
  - Unfair to compare software and hardware
  - The essence are inherent difficulties, accidents not inherent. Essence is interlocking concepts of data sets + their relationships + algorithms + invocations of functions



NSB -2

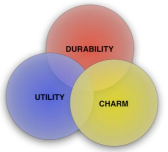
- Hard part of software is specification, design and testing of the conceptual construct not representing and testing its fidelity
- Inherent properties include:
  - Complexity - scaling software is not simply enlarging size of elements, rather increases # of different elements and their interaction
    - Math models will not work because you cannot abstract out the complexity - it is part of the essence
    - Complexity causes:
      - Difficulty communicating among team members
      - Difficulty in enumerating all possible states
      - Unvisualized states resulting in security trap doors
      - Difficulty in providing overview



NSB-3

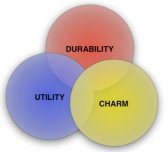
- Additional Inherent Properties:
  - Conformity - unlike physics with underlying laws, software reflects the arbitrary complexity of human institutions and systems and conformity to the interfaces of other systems
  - Changeability - software is constantly subjected to pressures to change:
    - Successful software is expanded
    - Exists longer than the machine vehicle for which it is written
    - Embedded in a changing cultural matrix of applications, machines, laws, organizations, …
  - Invisibility - hard to visualize in a single space

Penn CIS



NSB-4

- Breakthroughs solved many accidental difficulties
  - High level languages
  - Time sharing and now the network

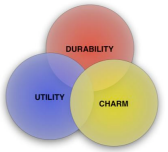


Exciting Products - Brooks

- Great Designers - Yes
- UNIX
- APL
- Pascal
- Modula
- Smalltalk
- Fortran

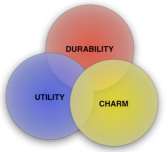
- Committees - No
- Cobol
- PL/1
- Algol
- MVS/370
- MS-DOS

Great designs come from great designers!



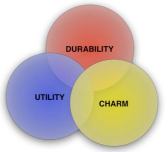
Great Designers

- Software is a creative process
- Ways to grow great designers (too little practiced these days):
  - Identify top designers early on
  - Career mentor them
  - Plot a career development path with apprenticeships and opportunities
  - Encourage peer interaction with other great designers



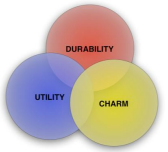
Larger Tidbits: Reuse

- Programmers reuse their own code (and friends)
- Mathematical software is heavily reused but it is a special case
- DeMarco - big expense to reuse, Yourdon quotes factor of 2 (1.5 to 3).
- Large class libraries, vocabulary is an impediment, simply too many things. Helpful aids:
  - Examples of composed products (not JAVADOC)
  - Learn vocabulary by use
- After all the critiques, no change in his premise - complexity is our business and it does limit us



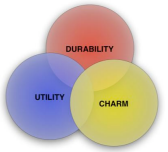
Chapter 18

- Summarizes all the chapters



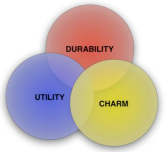
MMM after 20 years

- Really addresses how people in teams make things
- Central argument - conceptual integrity and the architect
  - System must be conceptually coherent to the single mind of the user, yet designed by many minds
  - The architect forms and owns the public mental model of the product
  - Separate the architecture, the definition of the product as perceived by the user(s) from its implementation, this boundary is within the design task, user facing vs implementation facing
- Recursion of architects - large projects have a master architect and the master architect partitions the system into subsystems with minimal, easily designed interfaces, each having its own architect
- Having a system architect is the single most important step to achieving conceptual integrity



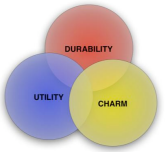
MMM - Retro 2

- Microsoft recommends build every night - (we do)
- Rapid Prototyping technique - Wizard of Oz
- Parnas was right about information hiding
  - Programmers are most effective if shielded from the innards of modules not their own (all in moderation)
  - More robust in a design for change (but …)
- How Mythical is the Man Month - hardly any projects succeed in 3/4ths of calculated time regardless of how many people are deployed



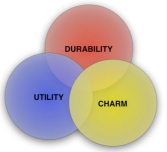
MMM-Retro 3

- How true is Brooks's Law : adding manpower to a late software project makes it later - most looked for remedies
  - Add manpower as early as possible
  - Focus on strategies for incorporating new folks
  - Brooks: none have addressed the added communication links
- People are (almost) everything
  - Quality of people and their organization and management are the major factors, *COCOMO* model quality of people is strongest factor
  - Managers must enhance creativity
  - Delegating power down

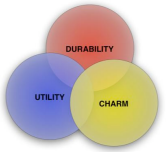


Software Engineering Concerns

- Remain the same:
  - How to design and build a set of programs into a system
  - How to design and build a program or a system into a robust, tested, documented and supported product
  - How to maintain intellectual control over complexity in large doses

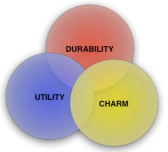


SOA



SOA: Service Oriented Architectures

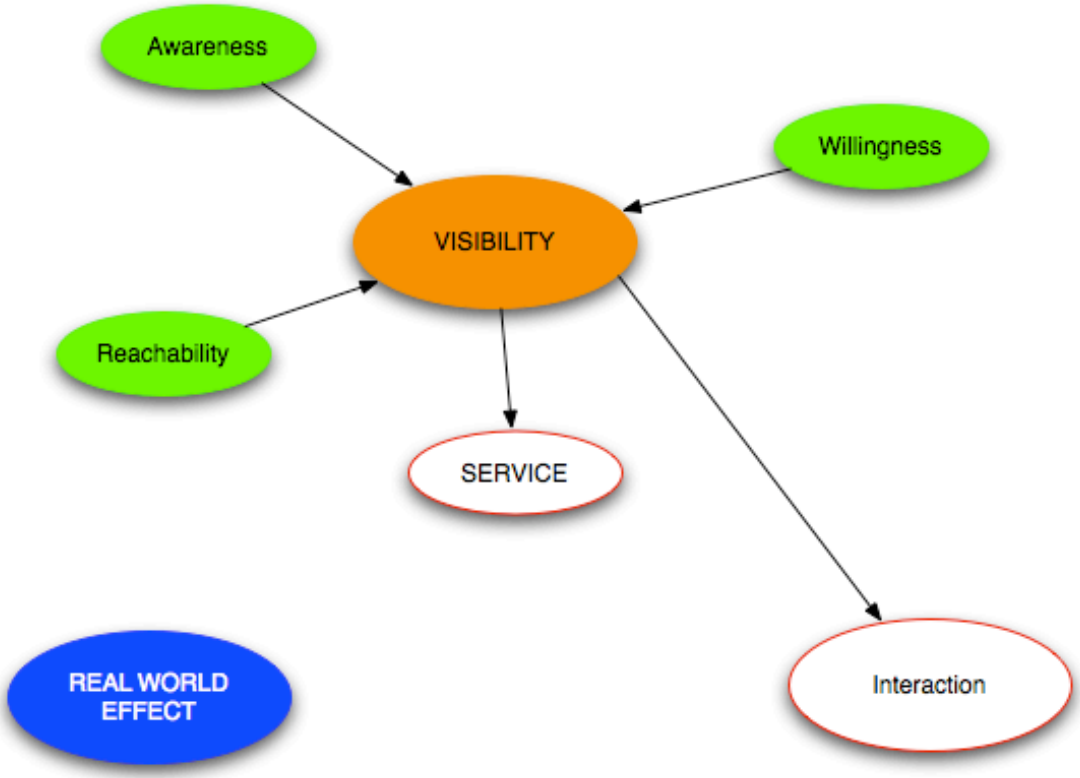
- (OASIS) SOA is a paradigm for organizing and utilizing distributed capabilities that may be under control of different ownership domains
  - Any given need may require the combining of numerous capabilities and any single capability may address multiple needs
- Visibility, interaction and effect are key concepts
  - Visibility - refers to the ability of those with needs and those with capabilities to see each other
  - Interaction is using the capability (service)
  - Results in real world effects -- an act not an object
- Service combines these ideas:
  - The capability to do work for another
  - The description and specification of the work offered
  - The actual offer to do work
  - Services are the mechanism that brings needs and capabilities together

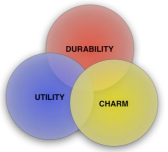


SOA -2

- SOA purpose is to assemble solutions that promote reuse, growth and interoperability (interfaces!)
- Service Participants:
  - Service providers
  - Service consumers
- Service description contains information necessary to interact with the service and describes the service in terms of inputs, outputs and associated semantics
- SOA is usually implemented using Web Services
- Central focus of SOA is the task or business function managed by delegation
- Service awareness can either be pushed or pulled

Visibility (OASIS, figure 4)





Service Interaction

- Information model is about information (duh) and requires consistent interpretation of strings and other tokens, requiring knowledge of the structure (syntax) and meaning (semantics).
  - Encryption service - information to decrypt/encrypt
  - Database service - requests to query or modify
- Behavior model - knowledge of actions permitted and process or time-oriented aspects of the interaction, described by actions on, responses to and timing dependencies (including sequences of actions).
  - Action model deals with the actions
  - Process model deals with temporal aspects including ordering
  - Note: when services are combined this gets funky and one discusses orchestration (a primary service invokes other services) choreography (services interacting with each other and maintaining state with no one service in control -- collaboration)

Service Description

- Information needed to use the service. SOA has a large amount of documentation and descriptions from a variety of customer perspectives. The information model is a key component of this.

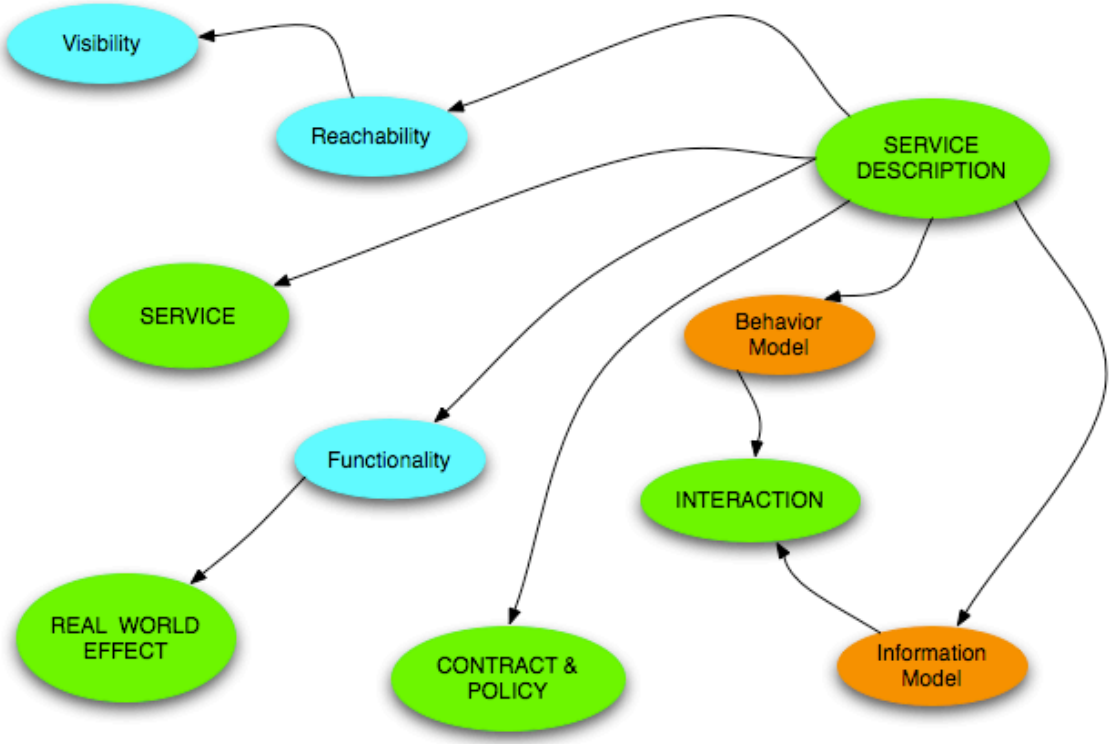
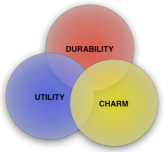


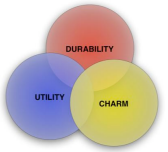
Figure 8 OASIS



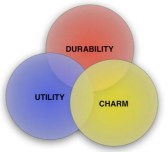
SOA Implementation (WEB)

- XML used extensively
  - XML schema (DTD -> XSD)
  - XSLT for transformation
  - XQuery
  - XPATH
  - SOAP vs. REST
- WSDL (Web Services Description Language) used for service description
- And much more later in Design if time

Penn CIS

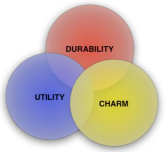


REAL TIME SOFTWARE ENGINEERING



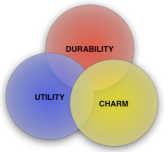
Real Time Systems

- Real time systems depends on the results of the system and the time the results are produced
  - Soft real time – operations are degraded if not produced on time
  - Hard real time – operations are incorrect if not produced on time
- One way of looking at it is as a stimulus and response system similar to software agents
  - Periodic stimuli – predictable intervals
  - Aperiodic stimuli – occur irregularly



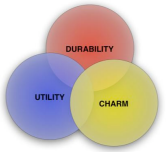
RT and Processes

- Usually designed as set of concurrent processes:
  - Sensor management
  - Computational
  - Actuator
- AGENTS!!



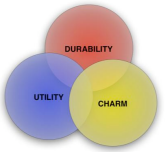
RT Design Process

- Identify stimuli(S) and related responses(R) that system handles
- Identify timing constraints for each S-R pair
- Choose an execution platform, hardware and software
  - timing constraints
  - Power constraints
  - Experience constraints
  - Cost constraints
- Aggregate S-R processing into concurrent processes – heuristic associate a process with each category of S-R
- For each S-R design algorithms – needed early to do performance studies
- Design a scheduling system to insure processes start on time



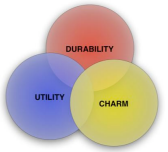
Process Coordination

- Ensures mutual exclusion to shared resources, tries to avoid deadlocks
  - Semaphores
    - Counting semaphore for pool of resources
  - Monitors
    - Intermediary between program processes and shared resources
- Check if they meet timing requirements
- May have to use implement in hardware
- Java and OO not the best



Real Time OS Components

- A real time clock
- An interrupt handler – manages aperiodic requests, NMI!
  - Interrupt level/priority
- A scheduler – selects processes
  - Preemptive – priority based
  - Non preemptive
- A resource manager
- A despatcher – starts process execution

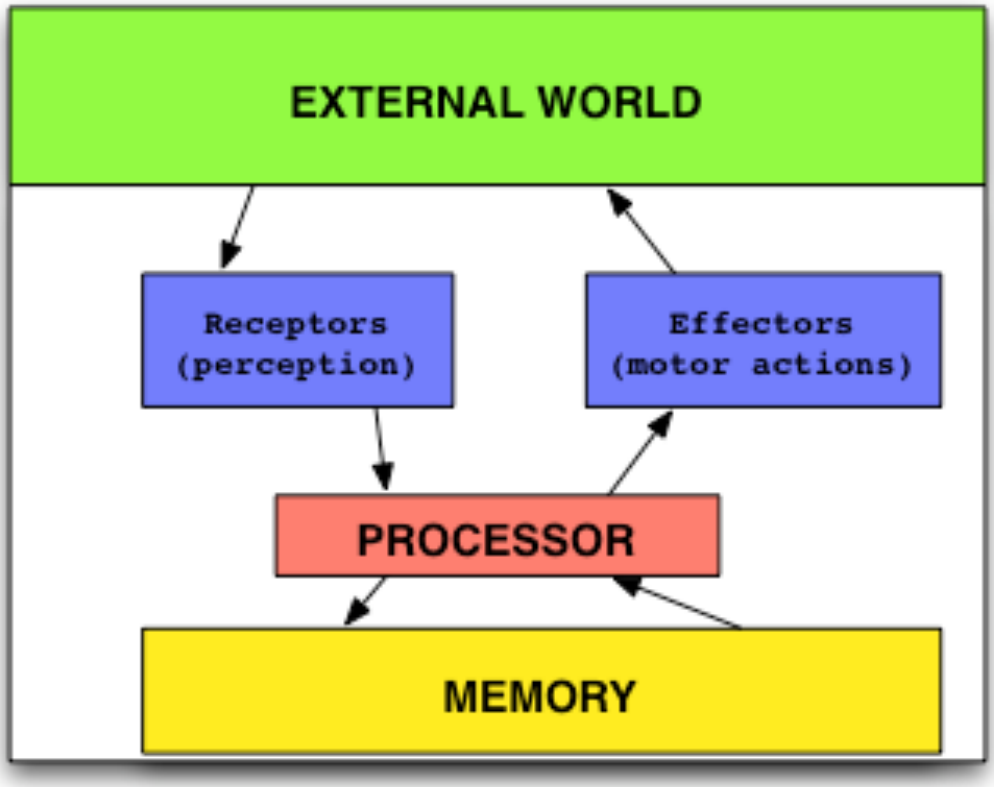


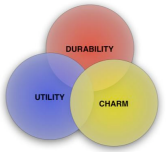
Monitor, Control, Data Acquisition

- Monitor
  - Usually soft real time – polling
  - Data acquisition
- Control
  - Actuators, thermostat
- Data Acquisition
  - Sampling is often necessary – analogous to polling

Human Information Processor

AI agent view

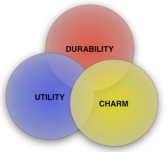




What's AI

- Artificial Intelligence not only tries to understand intelligence but also build intelligent entities (Russell & Norvig)

Cognition	Systems that think like humans	Systems that think rationally
Behavior	Systems that act like humans -> empirical science	Systems that act rationally -> math and engineering

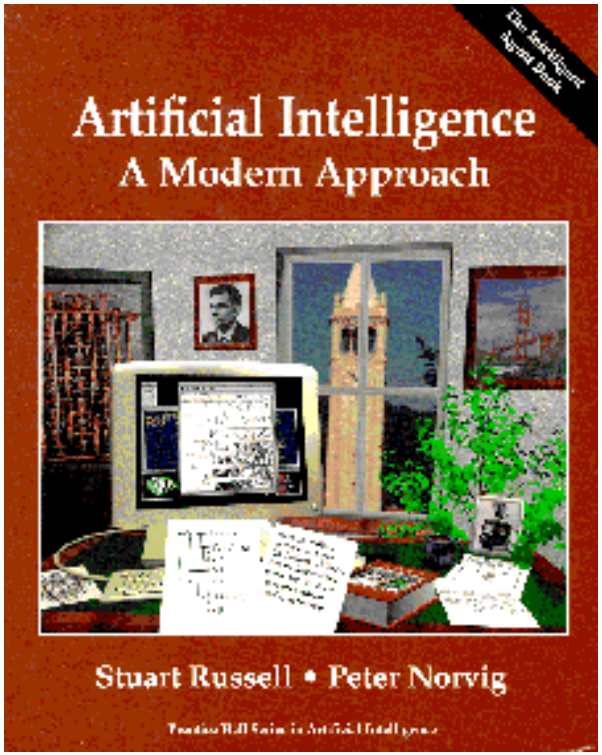


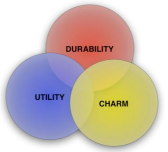
AI & Agents

- Agent: something that acts, operates under autonomous control, perceiving the environment, persisting over long time periods, adapting, being able to take on another's goals
- A rational agent achieves the best outcome or, given uncertainty, the best expected outcome
- Perfect rationality - always doing the right thing, is not feasible in complicated environments
- Limited rationality - acting appropriately when there is not enough time (or ability or feasibility) to do all the calculations one might like

Reference

Russell & Norvig Artificial Intelligence: A Modern Approach 2nd Edition, Prentice Hall, ISBN: 0137903952 (now green cover)

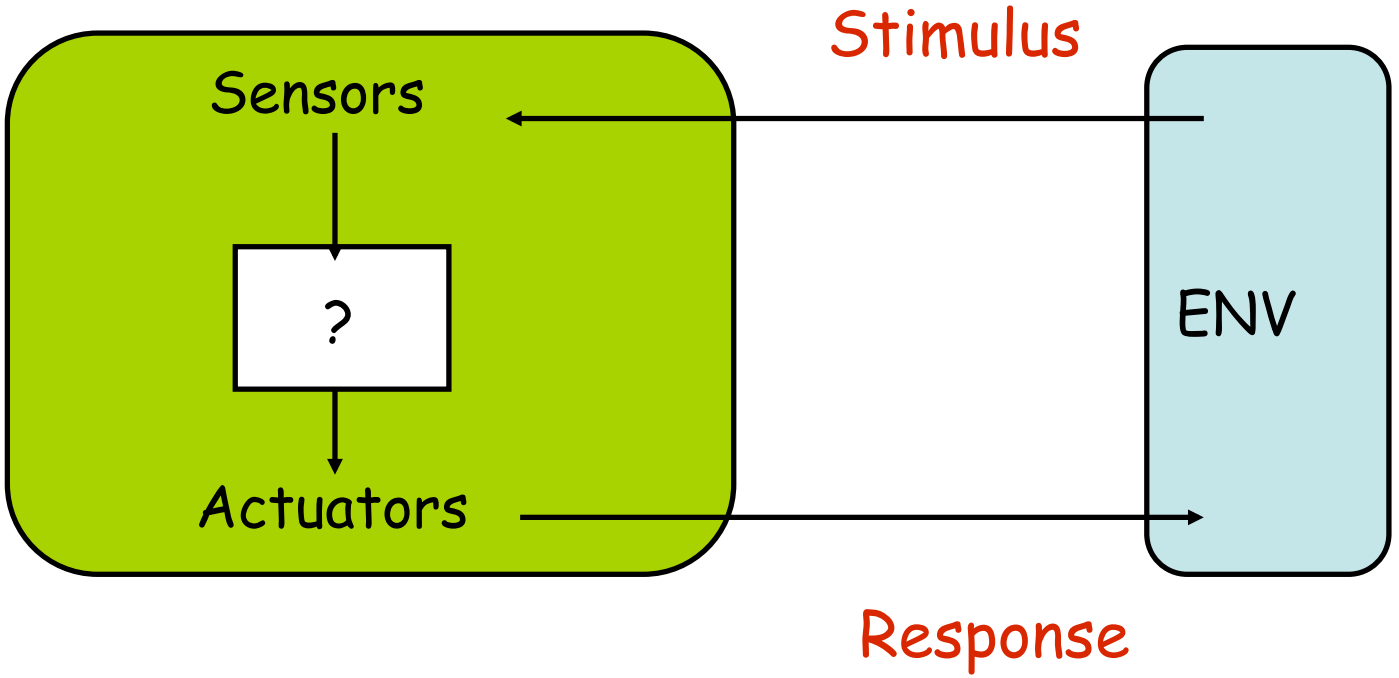


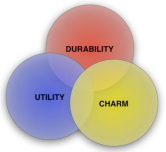


AI & Agents

- Agent: something that acts, operates under autonomous control, perceiving the environment, persisting over long time periods, adapting, being able to take on another's goals
- A rational agent achieves the best outcome or, given uncertainty, the best expected outcome
- Perfect rationality - always doing the right thing, is not feasible in complicated environments
- Limited rationality - acting appropriately when there is not enough time (or ability or feasibility) to do all the calculations one might like

Very Simplified Agent



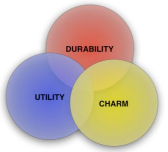


Classifying the Agent Task Environments

- Fully vs. Partially observable -can agent detect all relevant aspects?
- Deterministic vs. Stochastic - is next state of environment solely determined by current state and agent's actions
- Episodic vs. Sequential - does current decision affect future decisions?
- Static vs. Dynamic - can the environment change while the agent is deliberating?
- Discrete vs. continuous applies to state, time and to percepts & actions of the agents
- Single agent vs. Multi-agent - competitive or cooperative? Communication?
- **You're in trouble with partially observable, stochastic, sequential , dynamic, continuous and multi-agent!**

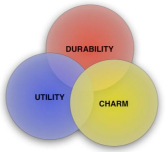
Types of Agents

- Simple Reflex Agents
  - Simple rule bases
- Model based reflex agent
  - Model state reflects part of the "percept history"
  - Information needed:
    - How world evolves independent of agent
    - How agent's actions affect world



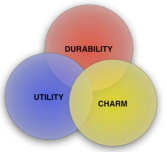
Types of Agents - 2

- Goal based agent
  - Agents use goals to select actions
  - Search and Planning devoted to action sequences achieving agent's goals
- Utility based
  - If one world state preferred to another then it has higher utility for agent
  - Utility functions are the measure
    - Enables tradeoffs on conflicting goals
    - Provides a way to measure likelihood of success against importance of goals

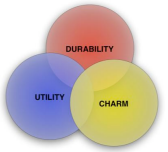


Learning and Agents

- Four components
  - Learning element
  - Performance element - selects external actions (The agent)
  - Critic - provides feedback
  - Problem generation - suggests actions leading to new informative experiences (exploration)

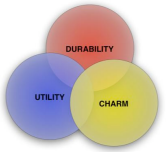


RELIABILITY



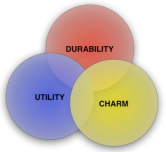
Reliability: vV quote

"The current-available software reliability models are limited in their immediate practical value." - p 629



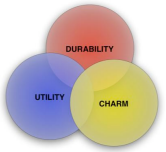
Bernstein - Fault tolerance

- Acknowledges that software has errors
- Goal: Mean Time To Failure, large and Mean Time To Repair, small -- making software available to users in the face of software errors
- In the 90's fault tolerance gave way to high availability, high reliability RAID based systems
  - Dominant vendors convinced folks this was enough
- "Software system development is frequently focused on performance and functional technical requirements and does not address the need for reliability or trustworthiness"
- Robustness deals with expected problems, fault tolerance with unexpected problems
- Trustworthy software is stable, does not crash with minor flaws and will shut down in an orderly fashion with major flaws



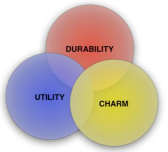
Early Approaches

- Initially mimicked hardware fault tolerance techniques
- N version concept in software mimics N-way redundant hardware in software
  - Each module is built with N different implementations, then each module submits its output to a "decider" that determines the correct answer.
  - Works if programs do not share similar failure modes
- Recovery Blocks - transactions are monitored and execution can be "rolled back"
  - Various things can be done to rolled back transaction including dropping it.
- But hardware fault tolerance is predicated on manufacturing defects



Software and Manufacturing

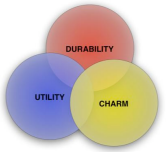
- Software faults are usually products of design shortcomings not manufacturing faults
  - Software manufacturing is configuration, …, i.e., reproduction
- Parnas' Undesired Events paper
- As Sha states:
  - Complexity begets faults
  - Some faults are easy to find and fix, others are Heisenbugs - a bug whose presence is affected by the act of observing it (intermittent symptoms, debuggers change situation)
  - Cannot spring for unlimited testing



HPC vs. HAC

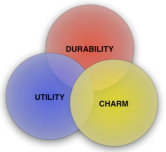
- High Assurance Controller
- Application Level - well understood classical, tradeoff performance for stability/simplicity
- System Software Level - high assurance, no frills OS
- Hardware level - well established, simple fault tolerant
- System Development and Maintenance - high assurance process
- Requirements Management- limits requirements to critical functions and essential services, stable and changes very slowly

- High Performance Controller
- Application Level - advanced technologies
- System Software Level- real time, dynamic and sophisticated development features
- Hardware Level - standard industrial hardware
- System Development and Maintenance - standard practices
- Requirements Management - emphasizes features and performance, requirements change relatively quickly



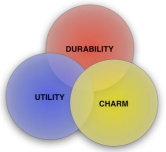
Fault Tolerance and Reliability

- Fault Tolerance is the ability of a software system to avoid executing a fault that causes a system to fail
- $R(t) = e^{-\lambda t}$ λ is proportional to software complexity, C, and inversely proportional to effort, E.
- $R(t) = e^{-kCt/E}$ and assuming duration t =1 and scaling constant k = 1, then $R(E,C) = e^{-C/E}$, the higher the complexity, the more effort.



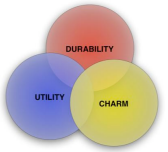
Reliability

- Can be improved by:
  - Investing in tools
  - Simplifying
  - Do more inspections and testing during development
    - Code reading, critique
    - Code reviews, check against requirements
    - Code inspection



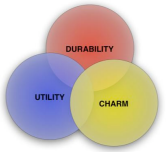
Constraints on Design

- Control free interfaces (low coupling)
- Software error recovery - exception handling throughout
  - Fault tolerance provides services consistent with spec after detecting fault - unknown problems
  - Exception handling contains and eliminates a known problem
- Recovery blocks
- Limit language features used
- Limit module size and initialize memory -- 300 to 500 instructions, smaller too many interfaces, larger too big for one person
- REUSE MODULES w/o CHANGE!



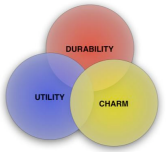
Moving to Simplicity - Refactoring

- Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure
  - Improving the design of the code after it has been rewritten
  - Make it work, make it work right, make it work better
  - Ultimate goal is to subtract code when adding a feature



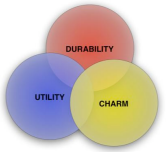
Increasing Reliability through Rejuvenation

- Rejuvenation - at periodic intervals, application is terminated and restarted at a known, clean state
- These are rejuvenation periods where everything is reinitialized
- A bug avoidance technique
- Usually measured in days and weeks - prevents memory leaks from getting the best of the system
- (systematic rebooting)



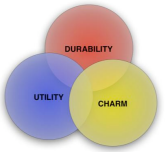
Effort Factors

- Hire good people, keep them
- Don't confuse vocational training with education
- Maximize effectiveness of staff, large changes (>100 instructions) and small changes (<5) are error prone
- Heuristics:
  - Design defensively, leave in debug mode
  - Stress test
  - Explain all anomalies uncovered, else provisional release
  - Hold arch reviews
- Classical engineering technique - stress a system until it breaks and guarantee it for much less than break point

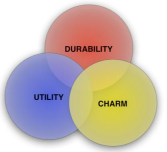


Robustness vs. FT

- Preventing a fault from becoming a failure - remember testing vocabulary
- It is fault tolerant iff:
  - Program can compute acceptable result even if it suffers from incorrect logic
  - Produces acceptable result even with corrupted data
- Robustness deals with expected problems, fault tolerance with unexpected problems

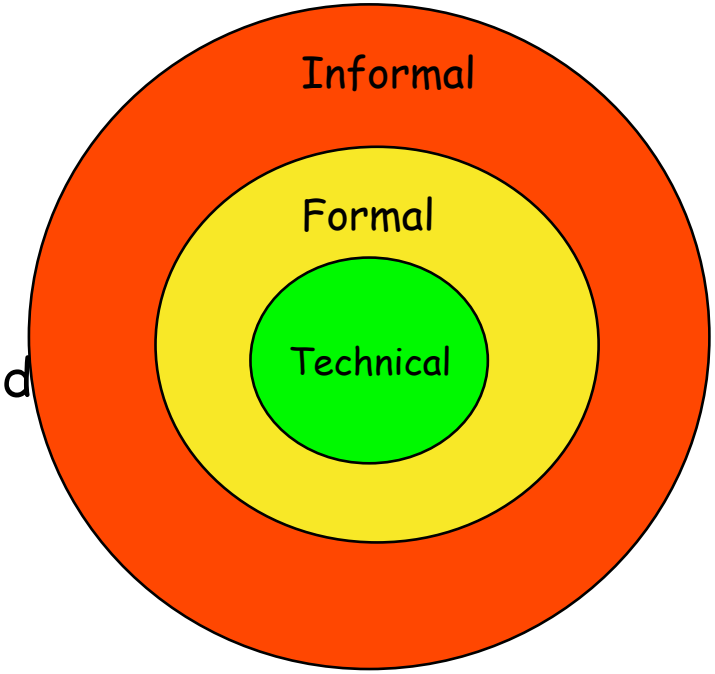


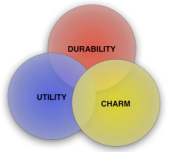
SECURITY



On Security

- Kaufman, et.al., "… how to communicate securely over an insecure medium"
- Garfinkel and Spafford(1991) "A computer is secure if you can depend on it and its software to behave as expected."
- Dhillon - "Coordination in three's"
- Schneier: prevention, detection, reaction





And Software has Issues

(McGraw2006)

- Connectivity- more software systems to attack, more attacks, greater risks
- Extensibility- evolving systems incrementally
- Complexity

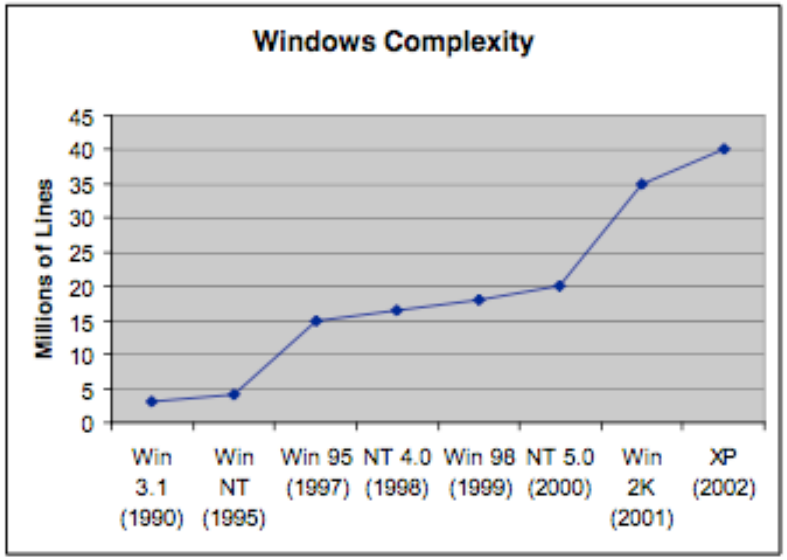


Figure 2: Growth of the Microsoft Operating System code base from 1990 to 2002.

Some Data

- 2002 - over 4,000
- 2003-4 - over 3,000

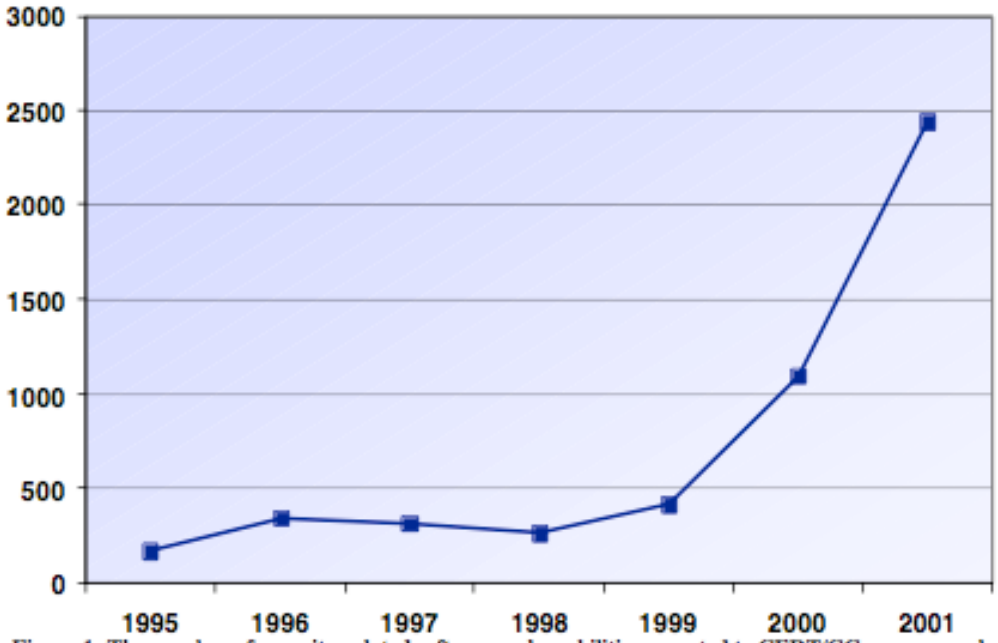
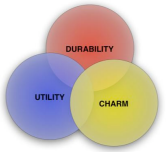


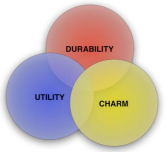
Figure 1: The number of security-related software vulnerabilities reported to CERT/CC over several years.



NIST Security Risk Assessment

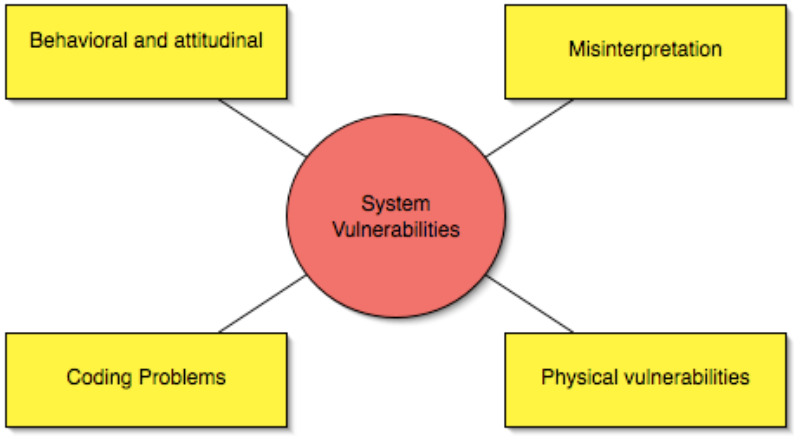
- System Characterization
- Threat Identification
- Vulnerability identification
- Control analysis
- Likelihood determination
- Impact analysis
- Risk determination
- Control recommendation
- Results documentation

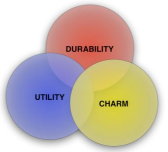
Risk is a function of the likelihood of a given threat resulting in certain vulnerabilities



Identify Risk

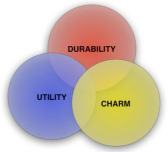
- Risks can be known, unknown and unknowable or known knowns, known unknowns (apply to this project), unknown unknowns :-)!
- SEI method of risk identification (and management) based on following assumptions:
  - Risks are often known by tech staff but poorly communicated
  - A repeatable method is necessary for risk management
  - Must cover all areas
  - Attitude must be non-judgmental and supportive so that controversial views can heard
  - Success or failure of the project can not be based solely on risk assessment





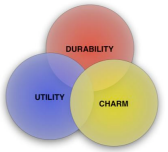
Analyze Risk

- Probability of risk, USAF Handbook categories are very low, low, medium, high and very high
- Impact of risk, USAF Handbook categories are negligible, marginal, critical and catastrophic
- Risks are rarely independent
- A matrix is used to determine overall risk for different categories (e.g., effort, performance, schedule, cost, support)



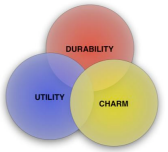
Sample Impact/Probability Matrix
(used to calculate overall risk)

Impact/Probability	V. High	High	Medium	Low	V. Low
Catastrophic	H	H	M	M	L
Critical	H	H	M	L	0
Marginal	M	M	L	0	0
Negligible	M	L	L	0	0



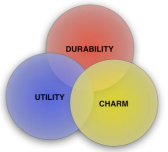
Some Considerations (McGraw)

- High (H) threat is highly motivated and capable with ineffective controls to contain it
- Medium (M) threat is motivated and capable but controls are in place that might thwart or slow it
- Low (L) threat not highly motivated or capable or controls are in place to thwart or minimize it



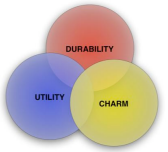
Plan for the Risks

- What can you do:
  - Mitigate impact by developing a contingency plan should risk occur and identify the trigger to initiate the contingency plan
  - Avoid the risk by changing something
  - Accept the risks and the consequences if it occurs, "do nothing"
  - Limit the risk by using controls
  - Transfer the risk, for example purchase insurance
  - Study the risk further so that you can decide on one of the above
- In addition:
  - Specify why risk is important
  - What info is need to track status of risk
  - Who is responsible for Risk Management and what is the cost, by identifying and managing the controls



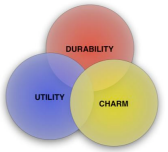
Risk Control Implementation (NIST)

- Assign priorities to actions, based on level of risk assigned during risk assessment
- Recommended control alternatives are evaluated
- Cost benefit analysis is conducted
- Controls selected based on cost benefit analysis
- Responsibilities are allocated (assigned)
- Safeguard implementation plan is developed
- Controls are implemented and assessment of residual risk is made
- Do this again at frequent intervals or when warranted



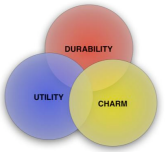
Risk Tracking and Control

- Track like everything else in the project monitoring status of risks and actions taken to address them. Appropriate risk metrics should be in place.
- Control, the risk process should be in place in the beginning, deviations to the plan should be corrected, triggering events should be handled and the process should be assessed for effectiveness.

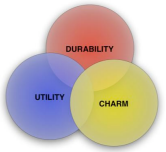


Some Risk Metrics (McGraw)

- Outstanding & identified risks by priority, type and subtype
- Risk mitigation by priority, percentage resolved and outstanding
- Risk discovery rate by priority and type
- Risk mitigation by priority and type, …

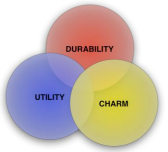


Some Risks



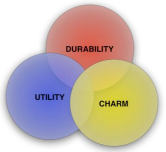
Script Kiddies

- Randomly search for a specific weakness and then exploit it
- Fact: your networks will be probed early and often
- General method:
  - Auto scan for a system they can exploit - time consuming
  - Use the system as a launching pad for other exploits (web site graffiti) - anonymity
    - Save this db for future use
  - Insert backdoors (easy and unnoticed access) and trojan horses making them undetectable - no log
  - DDoS a single user controlling tens to thousands of systems
- Unsophisticated (relatively) going for the easy kill
- Scan all systems at any time (automated)



Script Kiddies - 2

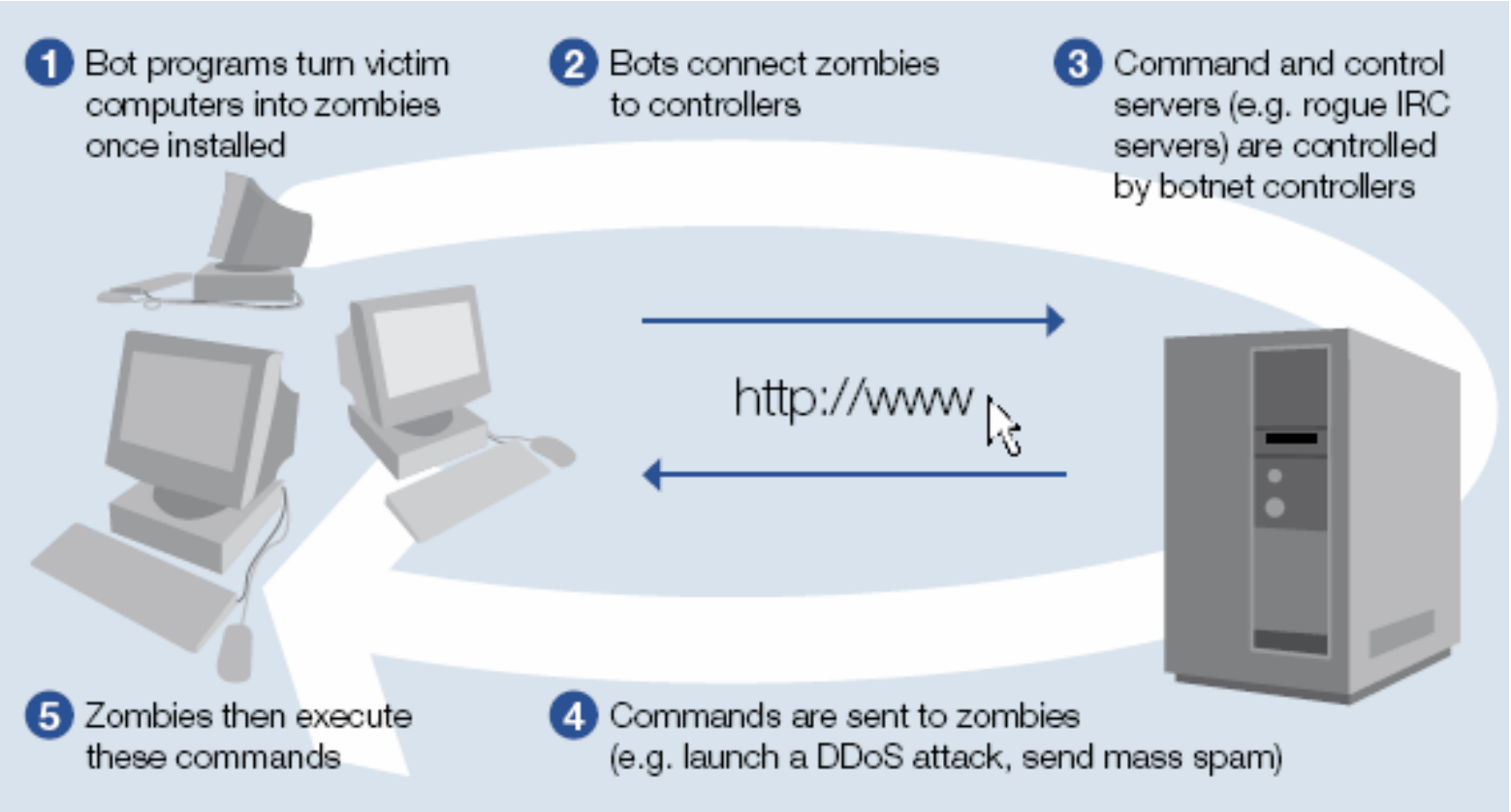
- Rootkit - a set of recompiled unix commands (ps, passwd, netstat) that hides evidence of an intruder, after installed a system is "rooted"
- Usually secure your system after they are on it to protect their prize
- Motives (dated, from "know your enemy" series):
  - Want to be k3wl, reputation (Open Source), leave your mark
  - DoS is big
  - Graffiti
  - Bringing on newbies
  - Lately used for IRC bouncing, bots, phishing scams

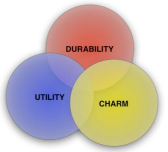


Botnets

- Network of compromised machines that can be controlled by an attacker - tens of thousands
- IRC based bots, zombies or drones, use IRC channel, 1to1 or 1tom communication
  - Bot joins a specific IRC channel and waits for instructions
- A botnet of 1000 bots can offer 100Mbit/s
  - BUSINESS MODEL
- Target specific ports - see paper
- Bots are constantly attempting to compromise more machines

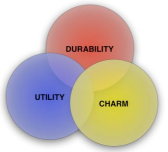
Building Botnets http://www.aic.gov.au/publications/tandi2/tandi333t.html





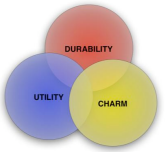
Uses of Botnets

- DDoS- Distributed Denial of Service - overwhelm victims network or computational resources
- Spamming -send out bulk mailings, also harvesting addresses from compromised machines for both spamming and phishing
- Sniffing traffic - retrieve sensitive information: usernames and passwds, control of other botnets
- Spreading new malware - bots beget bots
- Manipulating online polls and games
- Mass identity theft
- …



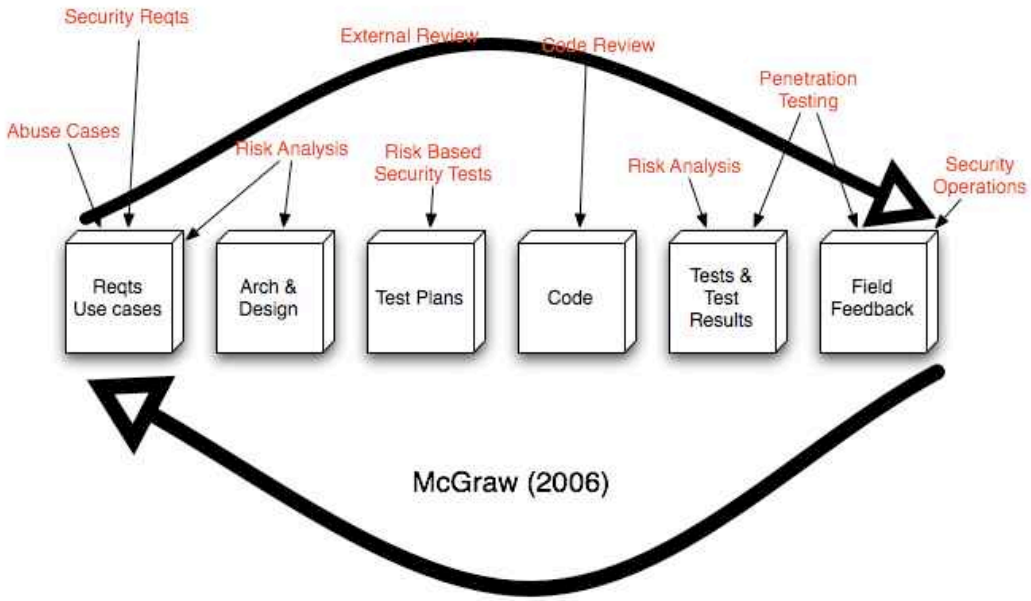
Remedies

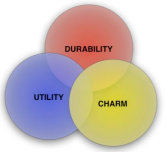
- Risk management and software development best proactices and of course:
  - Up to date patches
  - Highest security level (secures ports)
  - Secure ports
  - Virus protection
  - Detach from internet when not in use
  - Firewall
  - Intrusion Detection Systems
  - Back it up, emergency boot disk, system disks



SDLC AND SECURITY

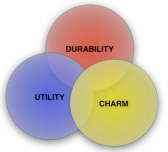
Software Development Best (Security) Practices ~ McGraw





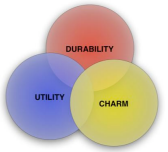
Code Review (rank 1)

- Input validation and Representation
- API abuse
- Security features
- Time and state
- Error handling
- Code quality
- Encapsulation
- ENVIRONMENT



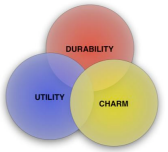
Architectural Risk Analysis (rank 2)

- Risk Management!!!
- Not an after thought but an important ilitie!
- McGraw's method:
  - Attack resistance analysis
  - Ambiguity analysis
  - Weakness analysis



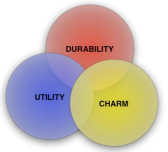
ARA-2

- Attack resistance analysis
  - Identify general flaws using design literature checklists (Microsoft STRIDE - Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege)
  - Map attack patterns - experience based
  - Identify risks in the arch based on above
  - Understand and demonstrate these are real attacks
- Ambiguity analysis - identify ambiguity and inconsistency in the arch (use experts) = vulnerabilities
- Weakness analysis - external software dependencies



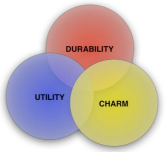
Penetration Testing (rank 3)

- Network penetration tests not the same as app or software penetration testing
- May represent a too little too late approach - late in life cycle
- (more in a bit)



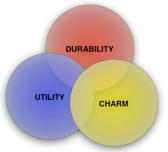
Risk Based Security Testing (rank 4)

- Grounded in systems architecture and attacker's mind set
- "A good threat is worth a thousand tests!" - Boris Beizer
- Outside - in AND Inside - out (way inside, focused on software guts)
- Like all testing it is both confidence building and destructive
  - Testing security mechanisms
  - Testing motivated by understanding and staging an attacker's approach



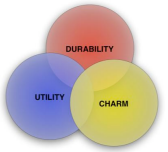
Abuse Cases (rank 5)

- Think like a bad person!
- Informed brain storming on threats and attacks
- Expose weakness - from all perspectives, inside and out!
- Anti-requirements - things you do not want software to do



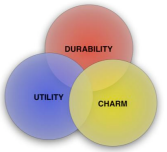
Some Attack Patterns

- Make client invisible - talk directly to serve mimicking a client
- Target programs that write to privileged OS resources
- Embedding scripts within scripts
- Argument injection
- …



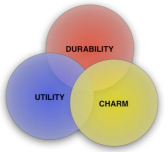
Security Requirements (rank 6)

- Understand and elicit
- Draw out potential attacks
- …



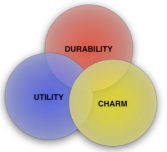
Security Operations (rank 7)

- It is part of O, A & M (Operations, Administration and Maintenance)
- If things change - review it
- Cannot make up for ignoring the other stuff - closing the barn door after …

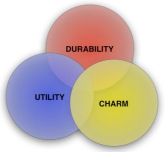


External Analysis (expert review - gtv)

- Throughout lifecycle
- Expert based
  - Red face test
  - From network to software



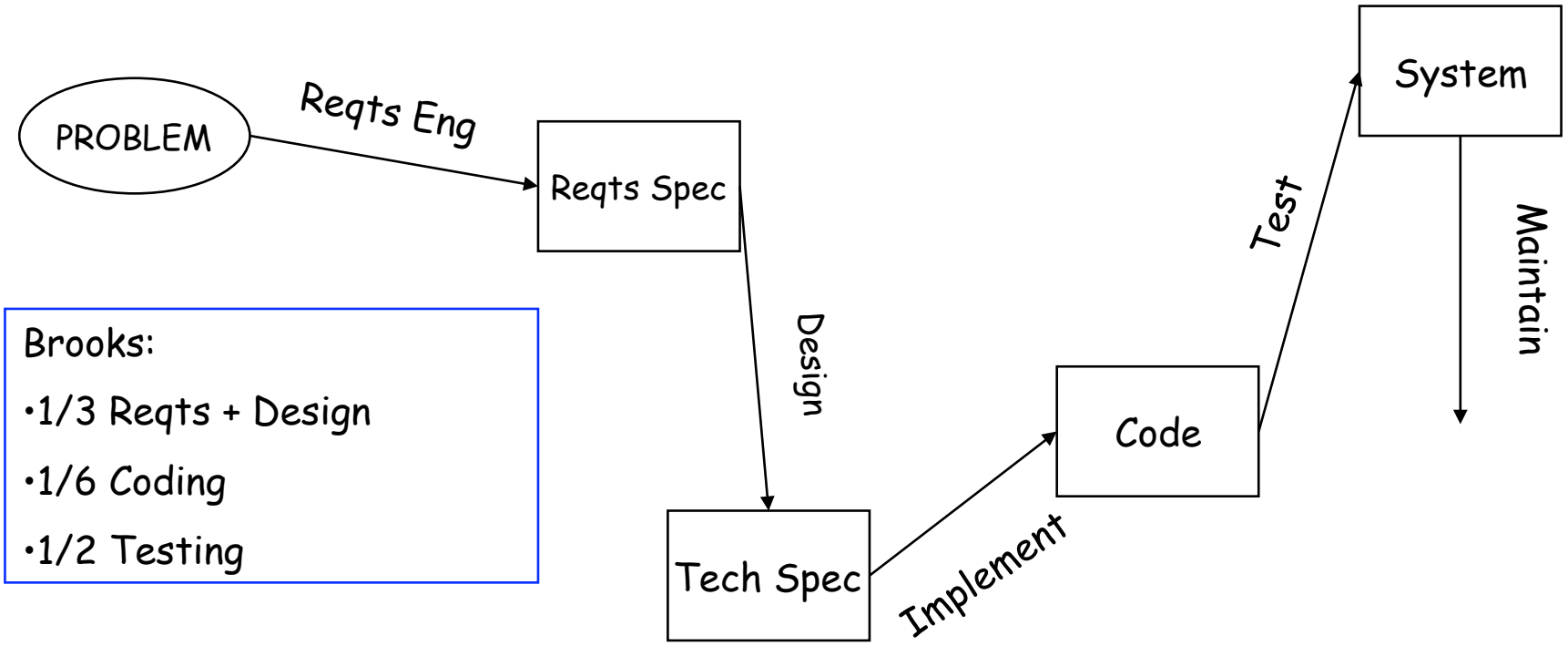
REVIEW!

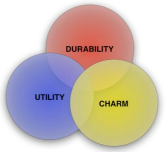


Software Engineering Knowledge

- SWEBOK, SoftWare Engineering Body Of Knowledge:
  - Software requirements analysis
  - Software design
  - Software construction
  - Software testing
  - Software maintenance
  - Software configuration management
  - Software quality analysis
  - Software engineering management
  - Software engineering infrastructure
  - Software engineering process

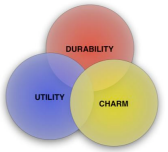
Simplified Model





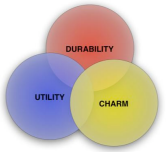
Class 1-3

- Software Engineering Overview
- Software Process Models
- CMM and the SEI
- Project Management
- Software Engineering micro and macro aspects
- Software Engineering Certification
- Requirements elicitation and representation
- Software project estimation
- Risk management
- Software Architecture
- Software Reviews



Class 4-end

- Design
- OO Design
- Quality
- Coding
- Testing
- Open Source
- Microsoft Development
- Anti-Patterns
- HCI
- Real Time Software Engineering
- Reliability
- SOA
- Security



References

- OASIS, "Reference Model for Service Oriented Architecture," http://www.oasis-open.org
- Kurose, J.F. and Ross, K.W. Computer Networking: 2nd Edition, Addison-Wesley, 2003. ISBN:0-201-97699-4
- G. Dhillon, Principles of Information Systems Security, Wiley, 2006, ISBN 0-471-45056-1
- Whittaker and Thompson, How to break software security, Addison-Wesley, 2004, ISBN: 0-321-19433-0
- Whittaker, How to break software a practical approach, Addison-Wesley, 2002.
- McGraw, G. Software Security, Addison-Wesley, 2006.
- Moriarity, C. Spin State, Bantam, 2003.
- http://project.honeynet.org
- Bernstein, L and Yuhas, C.M., Trustworthy Systems Through Quantitative Software Engineering, Wiley, 2005, ISBN 0-471-69691-9
- Cooling, J. Software Engineering for Real Time Systems, Addison-Wesley, 2003.
- Sha, L. Using simplicity to control complexity, IEEE Software, July/August, 2001, p 20-28