ECE/CS 541
Computer System Analysis:
Introduction & Syllabus

Mohammad A. Noureddine
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

Fall 2018

Learning Objectives

- Or what is this course about?

- At the start of the semester, you should have
  - Basic programming skills (C++, Python, etc.)
  - Basic understanding of probability theory (ECE313 or equivalent)

- At the end of the semester, you should be able to
  - Understand different system modeling approaches
    - Combinatorial methods, state-space methods, etc.
  - Understand different model analysis methods
    - Analytic/numeric methods, simulation
  - Understand the basics of discrete event simulation
  - Design simulation experiments and analyze their results
  - Gain hands-on experience with different modeling and analysis tools

What's in it for you?

- **For MCS/MEng students**

- Average base pay for a reliability engineer: $85,261
  - Google: $126,736
  - Apple: $116,631
  - Microsoft: $114,092
  - Intel: $108,343[1]

[1] Source: glassdoor.com

What's in it for you?

- **For MS/PhD students**

- Plenty of research opportunities
- A lot of companies interested in doing research in this area
  - Google, IBM, Microsoft
  - Strong presence at systems conferences
    - OSDI: https://www.usenix.org/conference/osdi18
    - NSDI: https://www.usenix.org/conference/nsdi19
    - Usenix ATC: https://www.usenix.org/conference/atc18
    - DSN, ICDCS, …

Course Resources

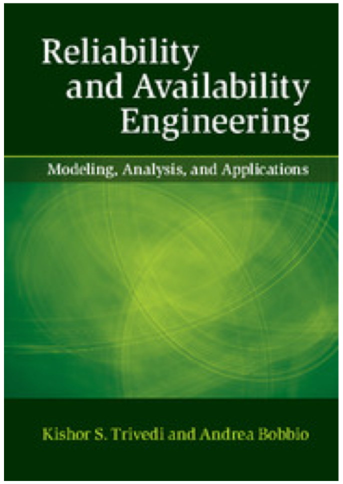
- Course website
  - https://courses.engr.illinois.edu/ece541/fa2018/
  - Updates, announcements, course schedule, homework, solutions, reading list, …
  - Authoritative source for deadlines, schedule, and exam times
  - Full syllabus: https://courses.engr.illinois.edu/ece541/fa2018/syllabus/
- Course Piazza page
  - Sign up at https://piazza.com/illinois/fall2018/ececs541
  - Go here to ask questions and start discussions
  - Use the forum to look for teammates for your project
- If assignment requires code submission, use Compass2g

Textbook

- No required textbook for the course

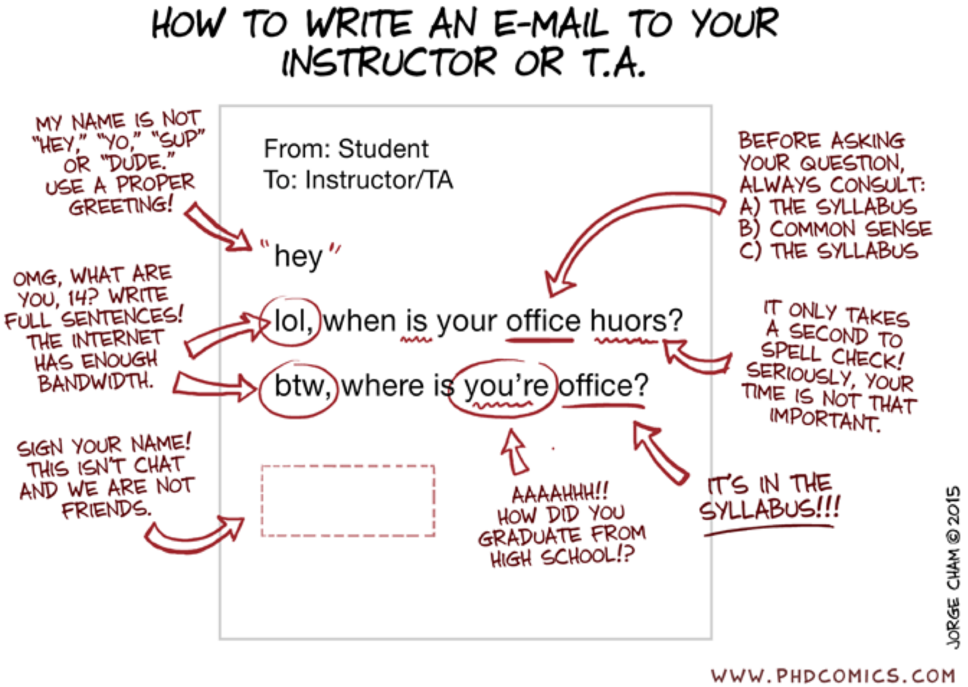
- We will post reading material to accompany each lecture whenever applicable
  - Book chapters
  - Conference papers
  - Journal papers

- Good, modernized reference
  - Kishor S. Trivedi and Andrea Bobbio, *Reliability and Availability Engineering,* Cambridge University Press, August 2017
    - **Available as an e-copy at the University library!**



Meet the Team

- **Instructor:** Mohammad A. Noureddine
  - Office: CSL 232 – Will overflow into CSL 231 if needed!
  - Office Hours: Tuesdays and Thursdays 11:00 am → 12:00 pm
- **Teaching Assistant:** Kartik Palani
  - Office: CSL 448
  - Office Hours: Wednesday and Friday, TBA



Grading Policy

- Semester Project: **35%**
- In-class midterm: **30%**
  - Test your understanding of theoretical class material
  - End of October
- Homeworks:
  - HW 0: **2%**
  - 4 to 5 Homework: **25%**
- In-class probability quiz: **5%**
  - Thursday September 20^{th}, first 30 minutes of class
  - Make sure you have done HW0 and HW1 individually
  - Check yourself before we start state space methods
- ~~Paper presentation: **5%**~~
- Class participation: **3%**

Class Participation

- Come to class and actively participate
  - Ask questions
  - Respond to questions
  - Share some of your outside knowledge and experience

- Contribute **good** discussion questions and answers on Piazza
  - Class topic follow-ups
  - Interesting articles, papers, events, etc.
  - Administrative/bug questions do not count!

- Come to office hours

- How to ask questions the smart way?
  - http://www.catb.org/esr/faqs/smart-questions.html

Late Policy

- You start with 1 credit for a late homework
  - This will give you 5 days beyond the due date

- After you use your credit, no late submissions will be allowed

- We would like to provide you with feedback as consistently as possible
  - We cannot do that if you are submitting late every time

Semester Project

- Chose a system of interest to you (or your research)
  - Computer Architecture
  - Networking protocols
  - Key management for public key cryptography
  - Social networks
  - Genomics
  - Robotics
  - Check out the Usenix Computer Failure Data Repository (CFDR) for ideas
    - https://www.usenix.org/cfdr
- Define a model of your chosen system using the tools and techniques presented in class (and maybe extensions of them)
- Chose an appropriate analysis method
  - Are analytic methods feasible? Is simulation feasible at scale?
- Implement your model (using the tool of your choice) and analyze its outputs

Project Breakdown

- Emphasis will be on the approach you adopt to study your system
  - Even if your results do not lead you to winning a Nobel prize!

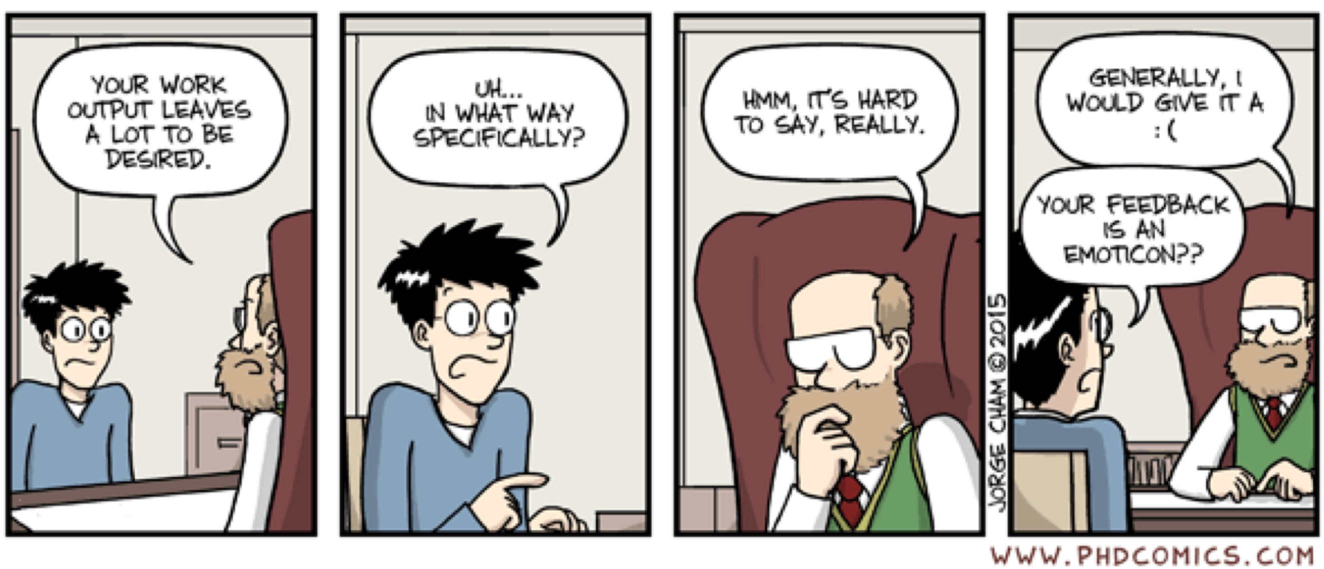
- Project will be done in teams of max 3 students
  - Use Piazza for finding teammates if you don't already have any

- Grade breakdown: **35%**
  - Project Proposal + literature review: **5%**
  - Project report (in paper format): **25%**
  - Project presentation (at the end of the semester): **10%**

- More information on this later on the course webpage
  - Proposal due around the end of September (first week of October) to allow ample time for feedback and discussion

Feedback Welcome Anytime (Please)!

- The goal of the course is to learn system modeling and analysis techniques
  - Success is measured by how close we get to this goal!
- We will do a midterm teaching evaluation
  - We will adjust the teaching and delivery based on your feedback
- Feedback on teaching, homeworks, exams, etc. during office hours also welcome.
- Want us to cover a specific topic/system/tool?
  - Let us know and we will try to arrange that!



The Rest of This Lecture

- Motivating examples
- Some definitions
  - What is a system or what are we analyzing?
  - Dependability and its components
  - Classification of faults, errors, and failures
  - Performance and its components
- The verification/validation dichotomy
- The validation tree
- Bring out the artist in you
  - Common pitfalls in computer system analysis
  - The "right" way!

Why are Analysis and Validation Important?

- Google Compute Engine's Service Level Agreement (SLA)

> *"During the Term of the Google Compute Engine License Agreement, Google Cloud Platform License Agreement, or Google Cloud Platform Reseller Agreement (as applicable, the "Agreement"), the **Covered Service will provide a Monthly Uptime Percentage to Customer of at least 99.99%** (the "Service Level Objective " or "SLO"). If Google does not meet the SLO, and if Customer meets its obligations under this SLA, Customer will be eligible to receive the Financial Credits described below"[1].*

- How does Google compute such an uptime number?
- How do we design our system to meet this objective?

[1]https://cloud.google.com/compute/sla

Evolution of Google's SLOs

- October 16, 2015:
  - Service Level Objective is 99.95%
  - Or *"three and a half nines"*
- November 4, 2016
  - SLO is 99.95%
- November 30, 2017
  - SLO is 99.95%

- April 13, 2018
  - SLO is 99.99%
  - Or *"four nines"*

- Why did it take Google three years to improve 0.04%?
  - How impactful are those 0.04%?

What does it mean for Google?

- If availability drops below 99.99%, Google will have to credit its customers for the downtime

- **<u>From Google's SLA:</u>**

Monthly Availability	Corresponding Downtime	% of monthly bill credited back to customer
99.00% - < 99.99%	4.3 minutes - < 7.2 hrs	10%
95.00% - < 99.00%	7.2 hrs – < 36 hrs	25%
< 95.00%	> 36 hrs	50%

- If Google's cloud service is down in a certain region for more than 4.3 minutes in a single month, you get 10% of your bill credited back!
  - As a reliability engineer, your goal is to design the system such that it is down for no more than 4.3 minutes in a month.

What does it mean for customers?

- How good is 99.99% availability?

- There are 8760 hours per year
  - Corresponds to ~53 minutes of downtime per year

- According to Gartner's reports[1], the cost of downtime is $300K per hour
  - So that's an estimate of $262.8K of losses every year

- <u>How much better is 99.99% compared to 99.95%?</u>
  - 99.95% uptime corresponds to 4.38 hours of downtime
  - An estimate of $1,314,000 losses every year

[1] https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/

Looking at it from Amazon's viewpoint

- During prime day, Amazon prime's website was down for 75 minutes
- Compare with the previous estimate of $300K per hour
  - Amazon's estimates were at **$1.2 million per minute**[1]

- Over the period of downtime, Amazon's losses hit an **estimated $90 million**

[1]https://techcrunch.com/2018/07/18/amazon-prime-day-outage-cost/

Safety Critical Systems

- Nuclear Power Plants
  - Failures can be catastrophic

- Intensive Care Units
  - Therac-25 killed at least 6 patients

- Airplanes
  - The Federal Aviation Administration (FAA) mandates that the probability of a catastrophic failure happening should be $< 10^{-9}$ per flight hour.

- **ECE541:**
  - How do we design systems that have such guarantees?
  - How do we evaluate whether existing designs satisfy these requirements?
  - Compare different designs with respect to certain properties.

What's a System?

- Many things, but in the context of this class, a collection of
  - hardware
  - networks
  - operating systems, and
  - application software

  that is intended to be dependable, secure, survivable or have predictable performance.

- Before learning how to validate such systems we must review
  1) Basic performance and dependability concepts and measures
  2) Fault/Error types
  3) Fault avoidance and tolerance techniques

What is Validation?

Definition:

Valid (*Dictionary Definition*)

- "Able to effect or accomplish what is designed or intended"

Basic notions:

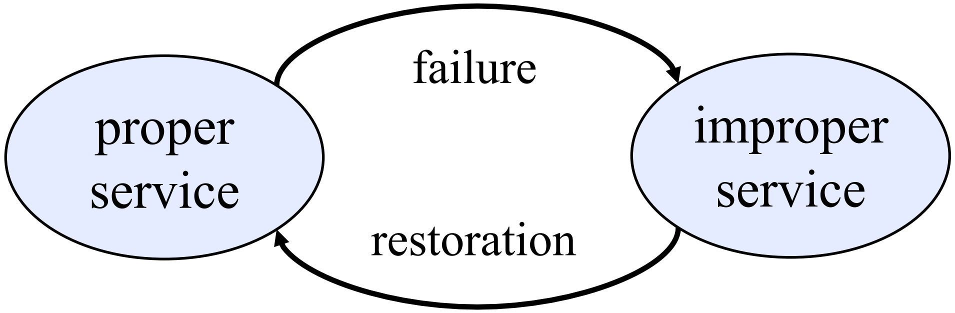
- *Specification* - A description of what a system is supposed to do.
- *Realization* - A description of what a system is and does.
- *Implementation* - A physical instance of the system.

Definition (for class):

Validation – we will often talk about "validating a model" to mean aligning the *specification* with the *realization*, and the *realization* with the *implementation*.

Properties of Interest: Dependability

- *Dependability* is the ability of a system to deliver a specified service.
- System service is classified as *proper* if it is delivered as specified; otherwise it is *improper*.
- System *failure* is a transition from proper to improper service.
- System *restoration* is a transition from improper to proper service.



⇒ The "properness" of service depends on the user's viewpoint!

Reference: J.C. Laprie (ed.), *Dependability: Basic Concepts and Terminology*, Springer-Verlag, 1992.

Examples of Specifications of Proper Service

- k out of N components are functioning.
- every bitcoin transaction will be eventually be confirmed
- every working processor can communicate with every other working processor.
- every message is delivered within t milliseconds from the time it is sent.
- all messages are delivered in the same order to all working processors.
- the system does not reach an unsafe state.
- 90% of all remote procedure calls return within x seconds with a correct result.
- 99.999% of all telephone calls are correctly routed.

⇒ Notion of "proper service" provides a specification by which to evaluate a system's dependability.

Dependability Concepts

- *Measures* - properties expected from a dependable system
  - Availability
  - Reliability
  - Safety
  - Confidentiality
  - Integrity
  - Maintainability
  - Coverage
- *Means* - methods to achieve dependability
  - Fault Avoidance
  - Fault Tolerance
  - Fault Removal
  - Dependability Assessment
- *Impairments* - causes of undependable operation
  - Faults
  - Errors
  - Failures

Dependability Measures: Availability

Availability - quantifies the alternation between deliveries of proper and improper service.

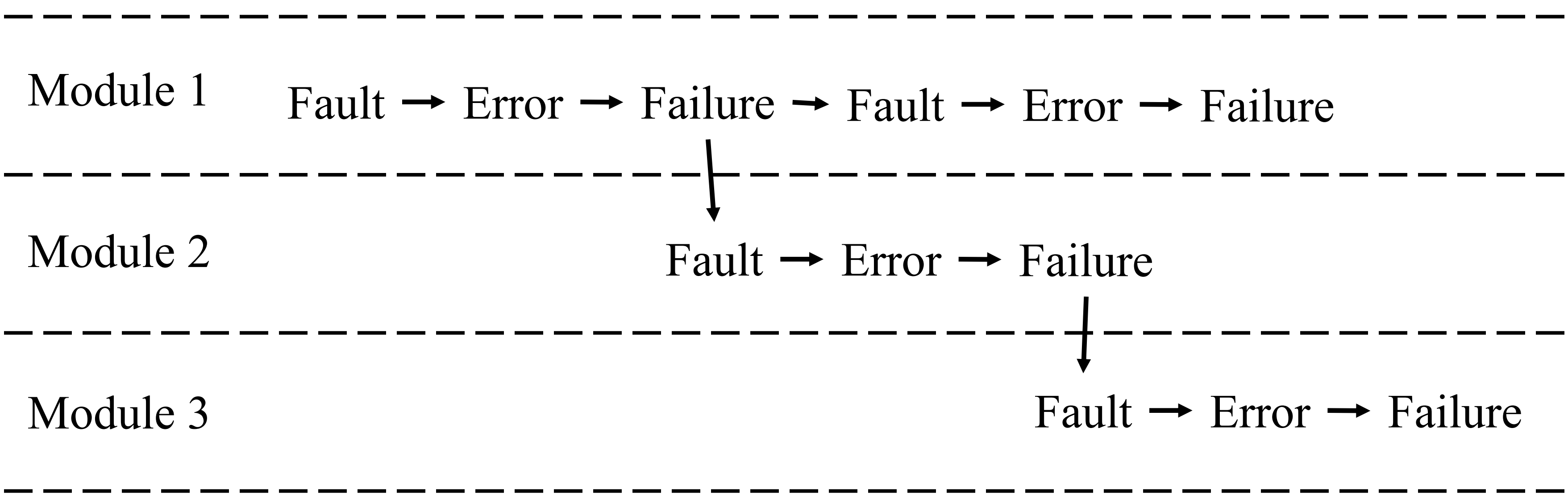
- $A(t)$ is 1 if service is proper at time t, 0 otherwise.
- $E[A(t)]$ (Expected value of $A(t)$) is the probability that service is proper at time t.
- $A(0,t)$ is the fraction of time the system delivers proper service during $[0,t]$.
- $E[A(0,t)]$ is the expected fraction of time service is proper during $[0,t]$.
- $P[A(0,t) > \alpha]$ $(0 \leq \alpha \leq 1)$ is the probability that service is proper more than $100\,\alpha\,\%$ of the time during $[0,t]$.
- $A(0,t)_{t\rightarrow\infty}$ is the fraction of time that service is proper in steady state.
- $E[A(0,t)_{t\rightarrow\infty}]$, $P[A(0,t)_{t\rightarrow\infty} > \alpha]$ as above.

Other Dependability Measures

- *Reliability* - a measure of the continuous delivery of service
  - $R(t)$ is the probability that a system delivers proper service throughout $[0,t]$.
  - If X represents the time to failure, then $R(t) = P[X > t] = 1 - P[X <= t]$
- *Safety* - a measure of the time to catastrophic failure
  - $S(t)$ is the probability that no catastrophic failures occur during $[0,t]$.
  - Analogous to reliability, but concerned with catastrophic failures.
- *Time to Failure* - measure of the time to failure from last restoration. (Expected value of this measure is referred to as *MTTF: Mean time to failure*.)
- *Maintainability* - measure of the time to restoration from last experienced failure. (Expected value of this measure is referred to as *MTTR: Mean time to repair*.)
- *Coverage* - the probability that, given a fault, the system can tolerate the fault and continue to deliver proper service.

Faults, Errors, and Failures can Cause Improper Service

- *Failure* - transition from proper to improper service
- *Error* - that part of system state that is liable to lead to subsequent failure
- *Fault* - the hypothesized cause of error(s), a defect within the system



Example

```
int main(int argc, char **argv)
{
        int a[5], *p = a, i = 0;
        while (i < 5) {
                *p++ = i++;
        }
        printf("I know pointer arithmetic!\n");
        printf("Here's the proof: the last element of a is %d\n", *p);

        // code to add  money to bank account
       ….

        return 0;
}
```

Example

```
int main(int argc, char **argv)
{
        int a[5], *p = a, i = 0;
        while (i < 5) {
                *p++ = i++;
        }
        printf("I know pointer arithmetic!\n");
        printf("Here's the proof: the last element of a is %d\n", *p);

        // code to add  money to bank account
       ….

        return 0;
}
```

Fault

Example

```
int main(int argc, char **argv)
{
        int a[5], *p = a, i = 0;
        while (i < 5) {
                *p++ = i++;
        }
        printf("I know pointer arithmetic!\n");
        printf("Here's the proof: the last element of a is %d\n", *p);

        // code to add  money to bank account
       ….

        return 0;
}
```

Fault

Error: Segmentation Fault

Example

```
int main(int argc, char **argv)
{
	int a[5], *p = a, i = 0;
	while (i < 5) {
		*p++ = i++;
	}
	printf("I know pointer arithmetic!\n");
	printf("Here's the proof: the last element of a is %d\n", *p);

	// code to add  money to bank account
	....

	return 0;
}
```

Fault

Error: Segmentation Fault

Failure: No money in account

Example

```
int main(int argc, char **argv)
{
        int a[5], *p = a, i = 0;
        while (i < 5) {
```

Fault


```
        return 0;
}
```

Failure: No money in account

Property of Interest: Performance

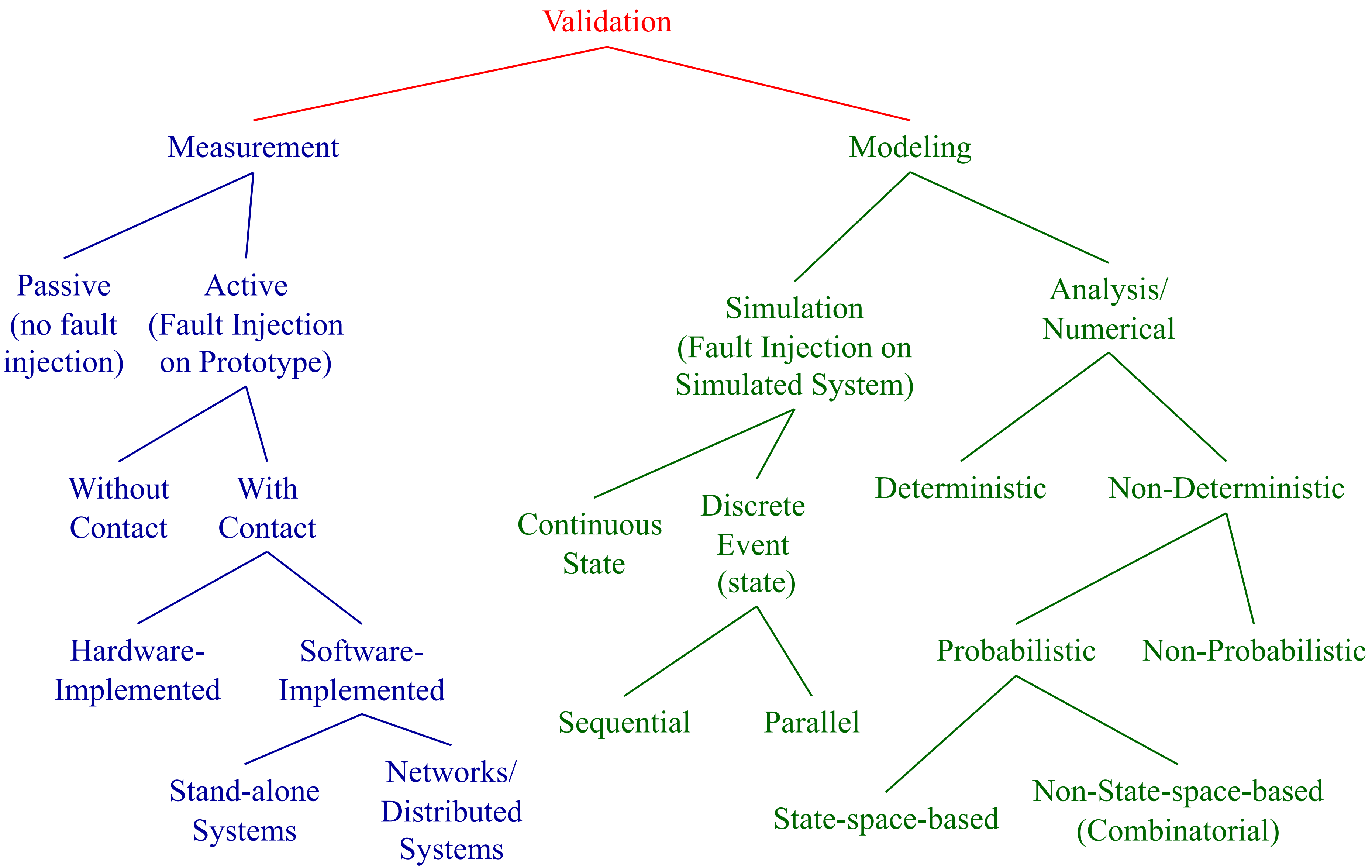
- *Performance* is how well a system performs, provides proper service
- Example (Generic) Performance Measures:
  - *throughput* -- the number of jobs processed per unit time
  - *response time* -- the time to process a specific job.
  - *capacity* -- the maximum number of jobs that may be processed per unit time
- Most practical performance measures are very application specific, and measure times to perform particular functions or, more generally, the probability distribution function of the time to perform a function.

A Combined Performance/Dependability Concept - Performability

- *Performability* quantifies how well a system performs, taking into account behavior due to the occurrence of faults.
- It generalizes the notion of dependability in two ways:
  – includes performance-related impairments to proper service.
  – considers multiple levels of service in specification, possibly an uncountable number.
- Performability measures are truly user-oriented, quantifying performance as perceived by users.

Original reference: J. F. Meyer, "On Evaluating the Performability of Degradable Computing Systems," *Proceedings of the 8th International Symposium on Fault-Tolerant Computing*, Toulouse, France, June 1978, pp. 44-49.

How is Validation Done?



Model Validation / Verification

Model validation (Am I building the correct model?)

– the process of making sure that the model you build is correct in the sense of accurately representing the system

Model verification (Am I building the model correctly?)

– the process of making sure that the model you're building captures your modeling intentions.

Models may be validated and verified using a number of methods:

- Modular design: test modules separately; interchange functionally similar modules,
- *N*-version models: high-level and detailed models should give similar results,
- Run simplified cases: e.g., one packet in the network,
- Tracing: examine one trajectory,
- Understand trends: understand the direction of the trends, and any discontinuities.

More Model Validation / Verification

- Models are frequently validated by three methods:
  - Measurements: measures on the model should match measures on the real system,
  - Theoretical results:
    - measure something to which you already know the answer,
    - e.g., throughput cannot exceed capacity,
  - Insight of experts:
    - Modeler expertise comes with experience,
    - Consult with people who understand the system.
- Validating a model is similar to validating software.
  - Design review: present the model design to a small committee that will critically review it.
  - "Code" review: someone else critically examines the model in detail.
  - Assertions: place assertions in the model to warn when things go wrong.
  - Black box/White box testing.

When Does Validation Take Place?

- *Specification* - Combinatorial modeling, Analytic/Numerical modeling
- *Design* - Analytic/Numerical modeling, Simulation modeling
- *Implementation* - Detailed Simulation modeling, Measurement, including fault injection
- *Operation* - Combinational modeling, Analytic/Numerical modeling, Detailed Simulation modeling, Measurement, including fault injection

Specification and Realization evolve throughout the lifecycle of a system.

Realization ultimately becomes an implementation.

Choosing Validation Techniques cont.

Criterion	Combinatorial	State-Space-Based	Simulation	Measurement
Stage	Any	Any	Any	Post-prototype
Time	Small	Medium	Medium	Varies
Tools	Formulae, tools	Languages & tools	Languages & tools	instrumentation
Accuracy	Low	Moderate	Moderate	high
Comparisons	Easy	Moderate	Moderate	Difficult
Cost	Low	Low/medium	Medium	High
Scalability	High	Low/medium	Medium	low

Some Rules of Thumb When Validating a System

- In general, always be suspicious of validation results... (e.g. don't accredit to them more accuracy than is warranted)
- Guideline: cross-check
  - Validate simulations with analytic models and measured data
  - Validate analytic models with simulations and measured data
  - Validate measured data with analytic models and simulations
- And, in particular, always
  - Evaluate "boundary cases" to which you know the answers
  - Make sure trends are as you expect, or understand why they are not

The "Art" of Performance and Dependability Validation

- Performance and Dependability validation is an art because:
  - There is no recipe for producing a good analysis,
  - The key is knowing how to abstract away unimportant details, while retaining important components and relationships,
  - This intuition only comes from experience,
  - Experience comes from making mistakes.

- There are many ways to make mistakes.

- You learn a tremendous amount about the system just by the act of modeling it and studying the model predictions

Doing it Right

- **Make all your assumptions explicit**:
  - Results from models, benchmarks, or fault-injection experiments are only as good as the assumptions that were made in obtaining them.
  - It's easy to forget assumptions if they are not recorded explicitly.
- Use the appropriate model solution or measurement technique:
  - Just because you have a hammer doesn't mean the world is a nail.
  - Fault injection and simulation, numerical/analytic, and combinatorial solutions all have their places.
- Keep social aspects in mind:
  - Dependability analysts almost always bring bad news.
  - Bearers of bad news are rarely welcomed.
  - In presentations, concentrate on results, not the process.

Doing it Right

- **Make all your assu**
  - Results from mo   experiments are only as good as the assu   them.
  - It’s easy to forg   ed explicitly.
- Use the appropriate   nique:
  - Just because you   orld is a nail.
  - Fault injection a   d combinatorial solutions all hav
- Keep social aspects
  - Dependability a   s.
  - Bearers of bad n
  - In presentations,   ss.

Course Overview

- Analytical/Numerical Methods
  - Review of Probability Theory
  - Combinatorial Methods
  - Fault Trees/Reliability Block Diagrams/Reliability Graphs
  - Review/Introduction to Stochastic Processes
  - Markov Modeling
  - Numerical/Tool Issues
  - High-Level Model Representations (Stochastic Petri Nets, Stochastic Activity Networks)
  - Numerical Issues in Markov Model Solution
  - Queuing Theory
  - Product-form Solutions of Queues

Course Overview, Cont

- Simulation
  - Simulation Basics
  - Modeling using the Scalable Simulation Framework
  - Simulation of Computer Systems and Network
  - Experimental Design
  - Validation and Verification
  - Variance Reduction
  - Simulation-based Optimization
- The Art of Computer System Assessment (Or how to get a good grade on the project)