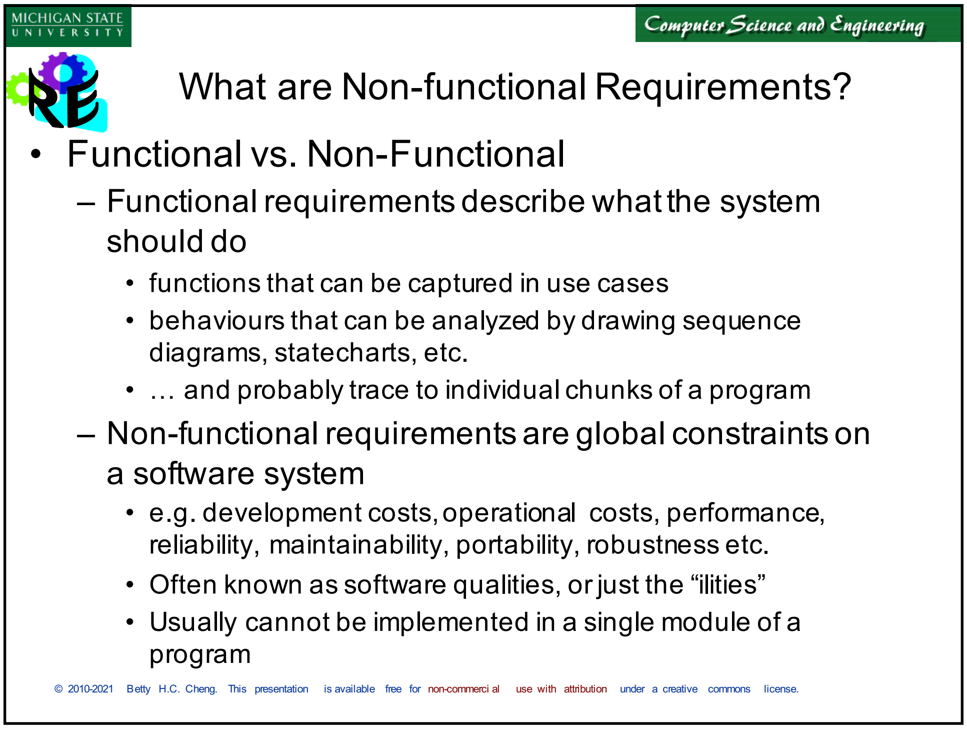
Non-Functional Requirements

Acknowledgements:
Steve Easterbrook

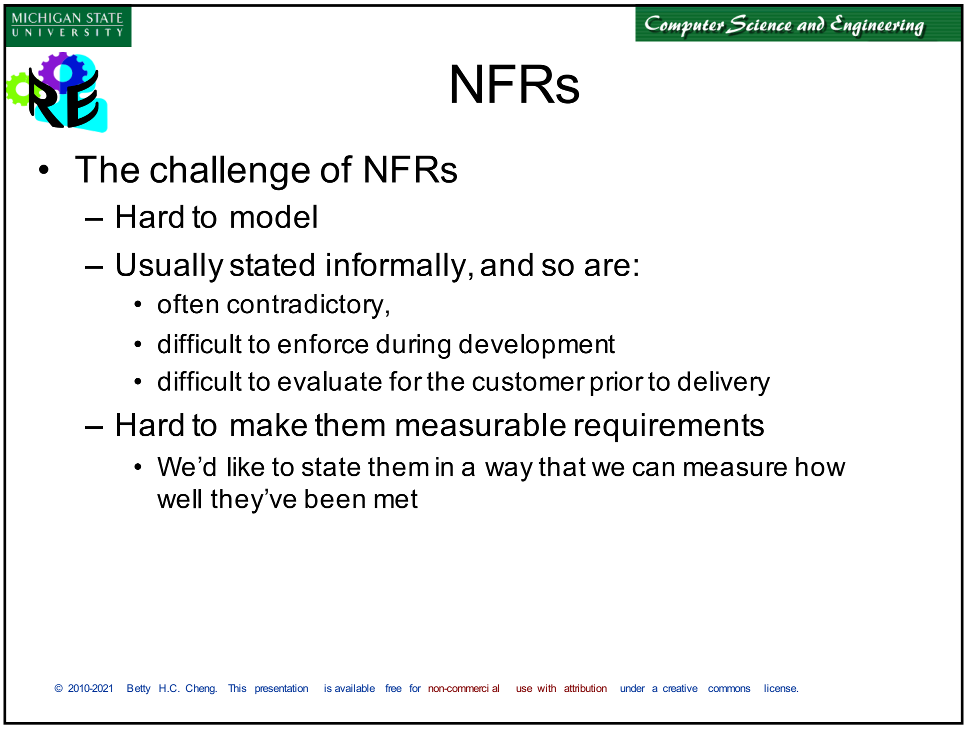
Non-Functional Requirements (NFRs)

- Definitions
  - Quality criteria; metrics
  - Example NFRs
- Product-oriented Software Qualities
  - Making quality criteria specific
  - Catalogues of NFRs
  - Example: Reliability
- Process-oriented Software Qualities
  - Softgoal analysis for design tradeoffs



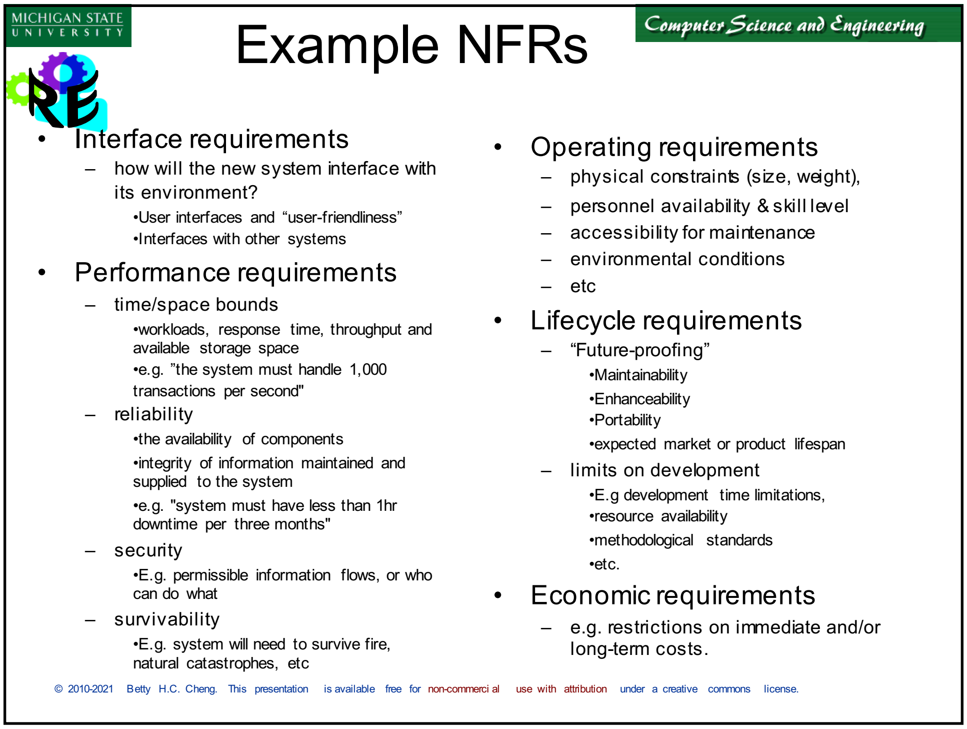
What are Non-functional Requirements?

- Functional vs. Non-Functional
  - Functional requirements describe what the system should do
    - functions that can be captured in use cases
    - behaviours that can be analyzed by drawing sequence diagrams, statecharts, etc.
    - … and probably trace to individual chunks of a program
  - Non-functional requirements are global constraints on a software system
    - e.g. development costs, operational costs, performance, reliability, maintainability, portability, robustness etc.
    - Often known as software qualities, or just the "ilities"
    - Usually cannot be implemented in a single module of a program



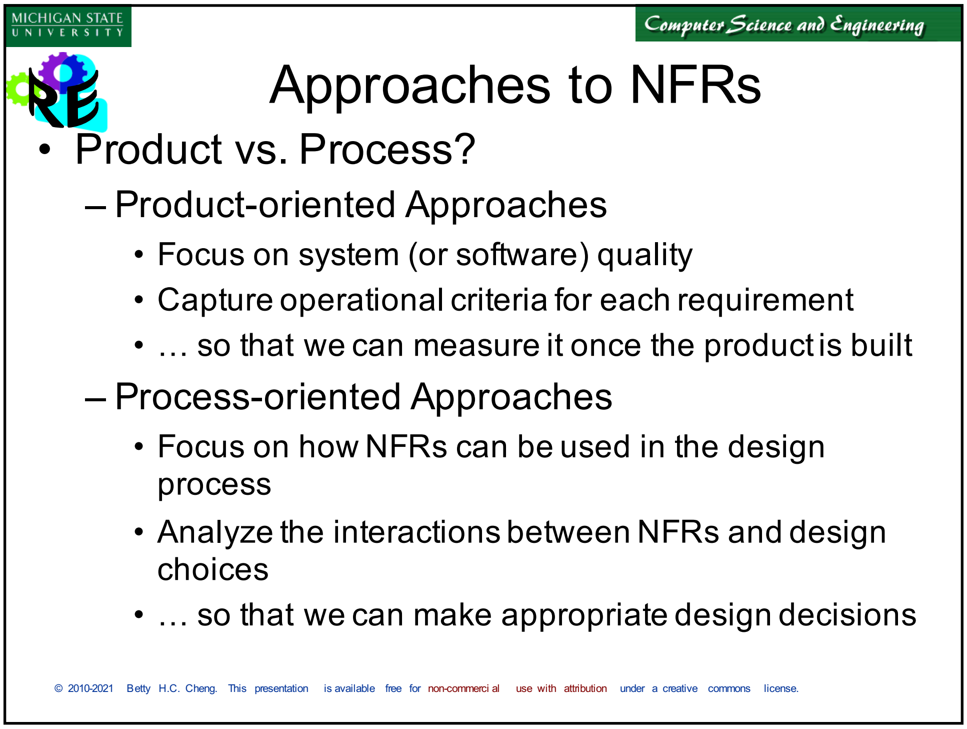
NFRs

- The challenge of NFRs
  - Hard to model
  - Usually stated informally, and so are:
    - often contradictory,
    - difficult to enforce during development
    - difficult to evaluate for the customer prior to delivery
  - Hard to make them measurable requirements
    - We'd like to state them in a way that we can measure how well they've been met



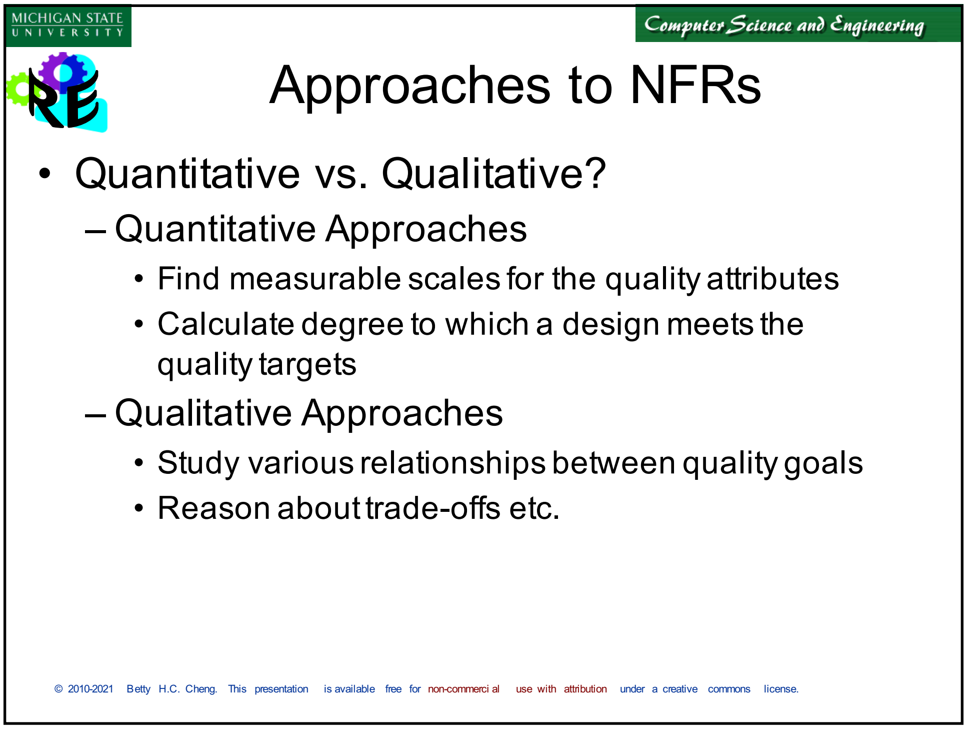
Example NFRs

- Interface requirements
  - how will the new system interface with its environment?
    - User interfaces and "user-friendliness"
    - Interfaces with other systems
- Performance requirements
  - time/space bounds
    - workloads, response time, throughput and available storage space
    - e.g. "the system must handle 1,000 transactions per second"
  - reliability
    - the availability of components
    - integrity of information maintained and supplied to the system
    - e.g. "system must have less than 1hr downtime per three months"
  - security
    - E.g. permissible information flows, or who can do what
  - survivability
    - E.g. system will need to survive fire, natural catastrophes, etc
- Operating requirements
  - physical constraints (size, weight),
  - personnel availability & skill level
  - accessibility for maintenance
  - environmental conditions
  - etc
- Lifecycle requirements
  - "Future-proofing"
    - Maintainability
    - Enhanceability
    - Portability
    - expected market or product lifespan
  - limits on development
    - E.g development time limitations,
    - resource availability
    - methodological standards
    - etc.
- Economic requirements
  - e.g. restrictions on immediate and/or long-term costs.



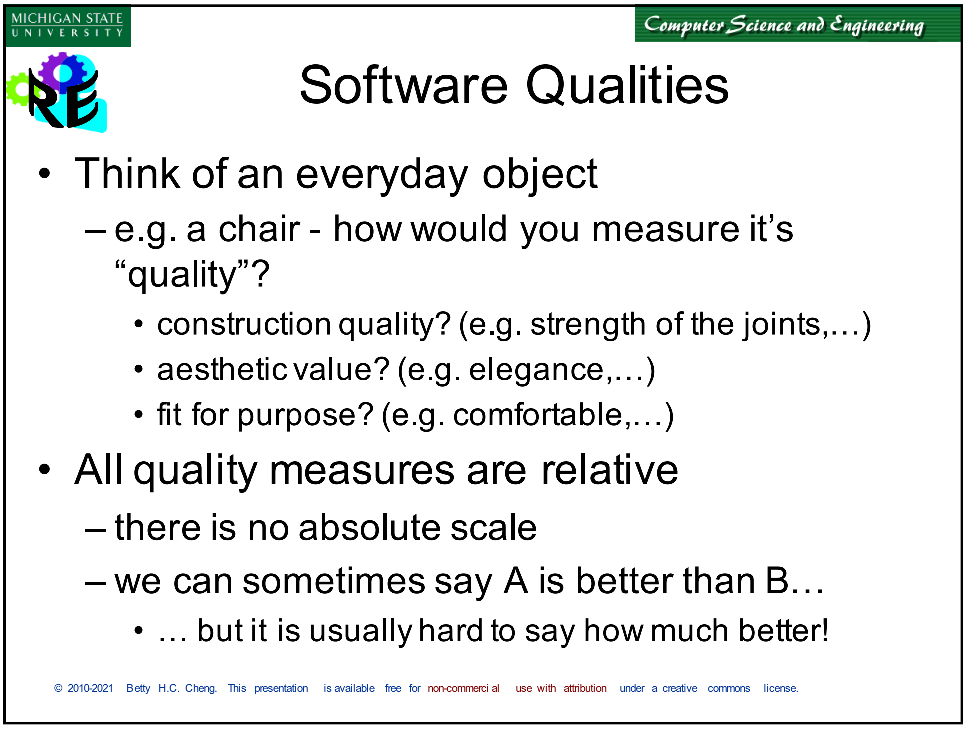
Approaches to NFRs

- Product vs. Process?
  - Product-oriented Approaches
    - Focus on system (or software) quality
    - Capture operational criteria for each requirement
    - … so that we can measure it once the product is built
  - Process-oriented Approaches
    - Focus on how NFRs can be used in the design process
    - Analyze the interactions between NFRs and design choices
    - … so that we can make appropriate design decisions



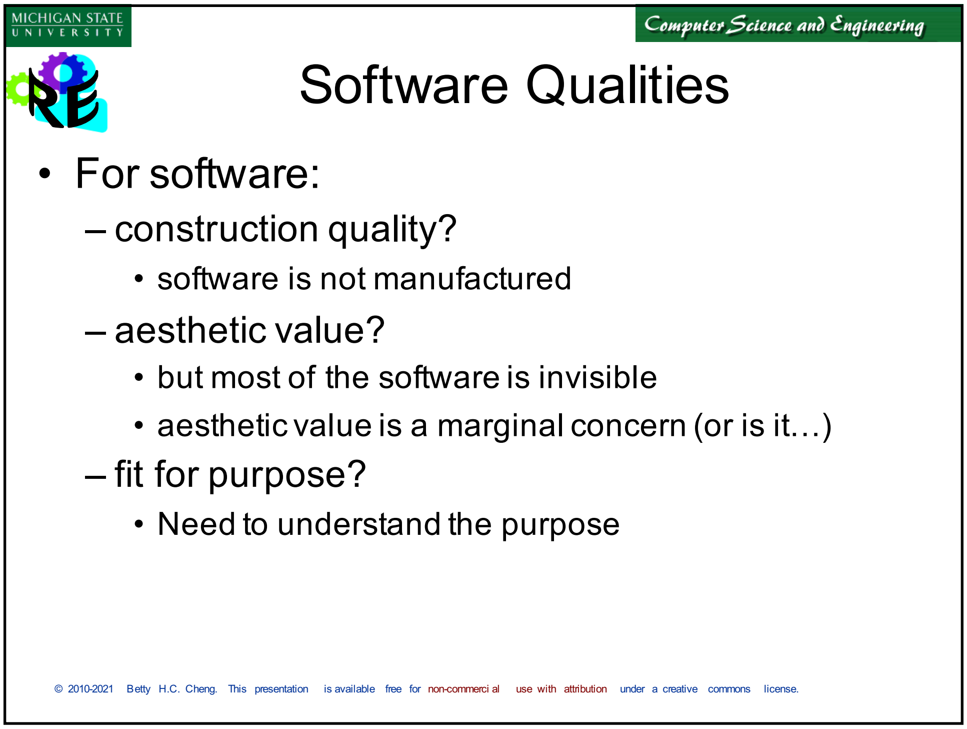
Approaches to NFRs

- Quantitative vs. Qualitative?
  - Quantitative Approaches
    - Find measurable scales for the quality attributes
    - Calculate degree to which a design meets the quality targets
  - Qualitative Approaches
    - Study various relationships between quality goals
    - Reason about trade-offs etc.



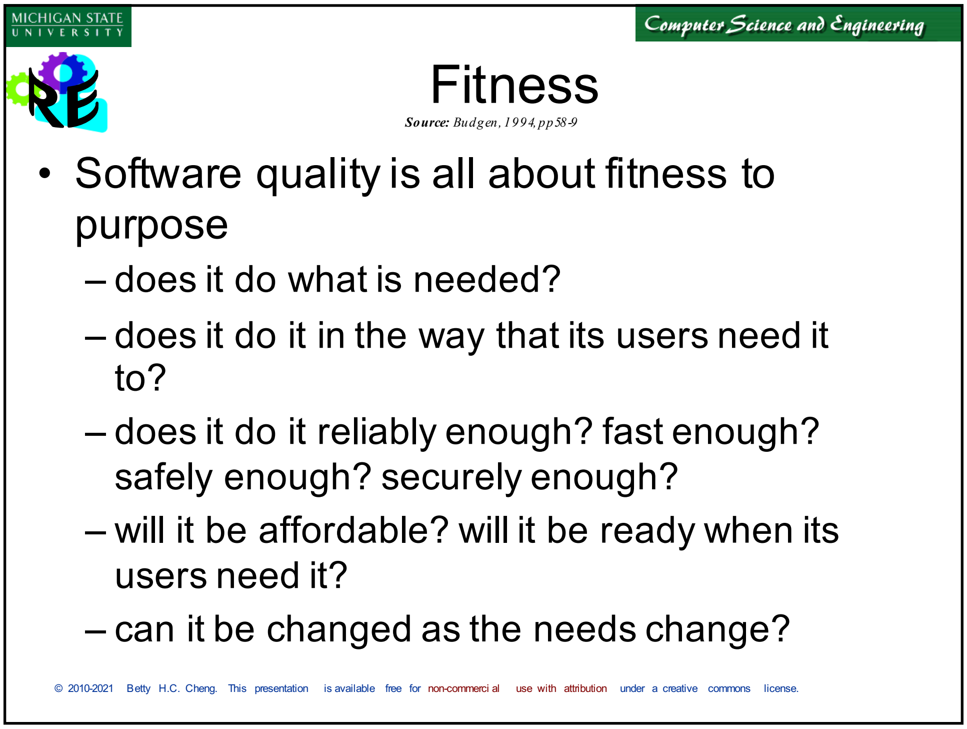
Software Qualities

- Think of an everyday object
  - e.g. a chair - how would you measure it's "quality"?
    - construction quality? (e.g. strength of the joints,…)
    - aesthetic value? (e.g. elegance,…)
    - fit for purpose? (e.g. comfortable,…)
- All quality measures are relative
  - there is no absolute scale
  - we can sometimes say A is better than B…
    - … but it is usually hard to say how much better!



Software Qualities

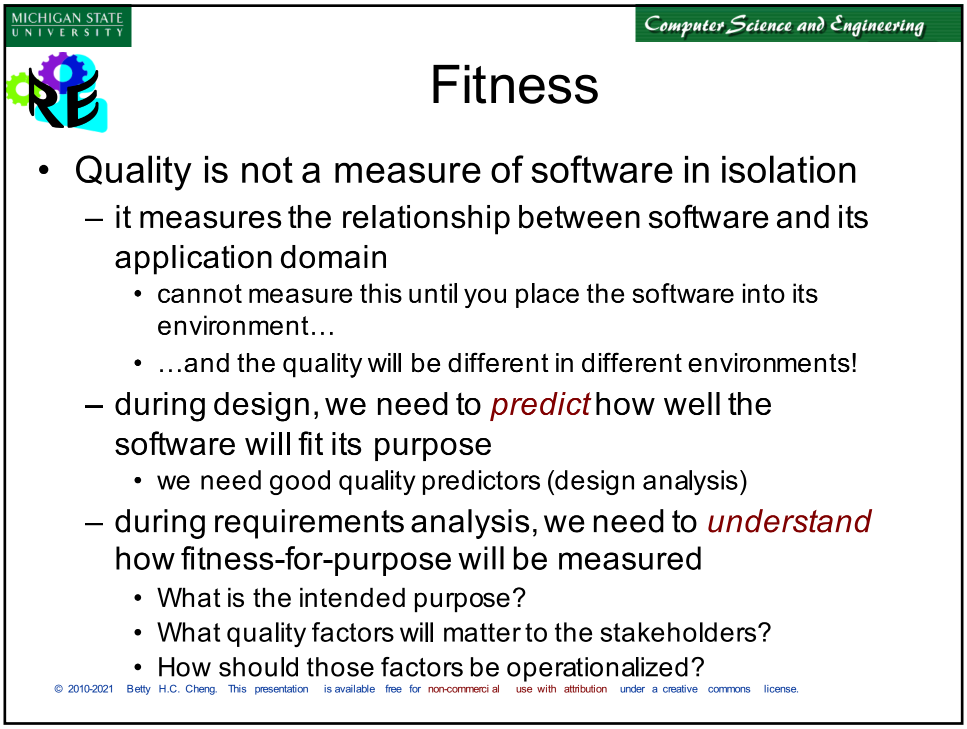
- For software:
  - construction quality?
    - software is not manufactured
  - aesthetic value?
    - but most of the software is invisible
    - aesthetic value is a marginal concern (or is it…)
  - fit for purpose?
    - Need to understand the purpose



Fitness

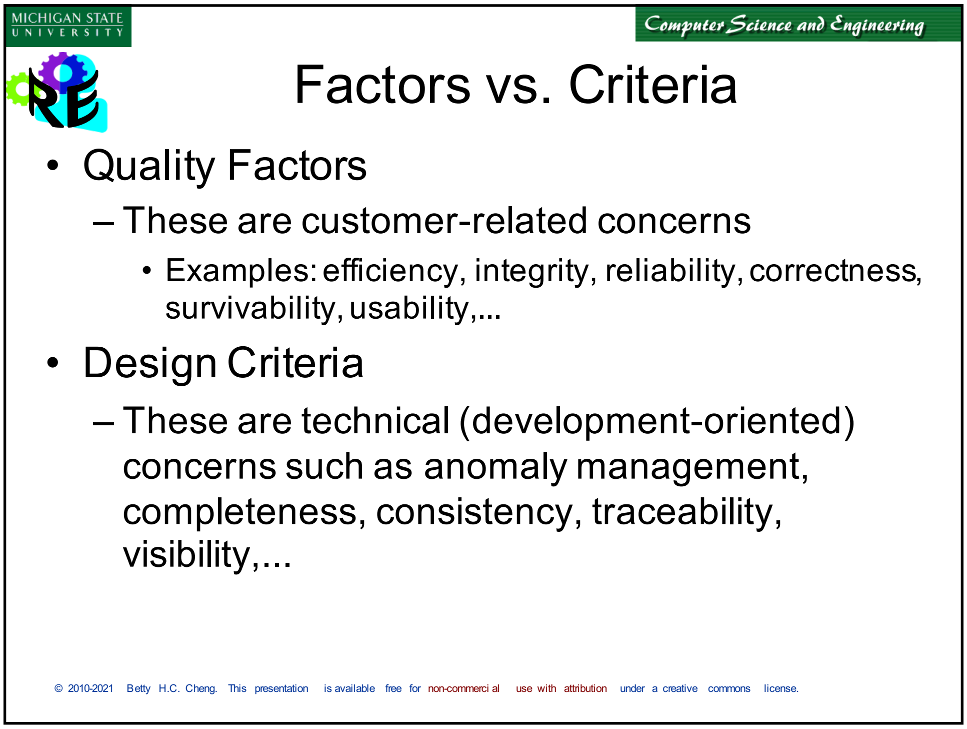
Source: *Budgen, 1994, pp 58-9*

- Software quality is all about fitness to purpose
  - does it do what is needed?
  - does it do it in the way that its users need it to?
  - does it do it reliably enough? fast enough? safely enough? securely enough?
  - will it be affordable? will it be ready when its users need it?
  - can it be changed as the needs change?



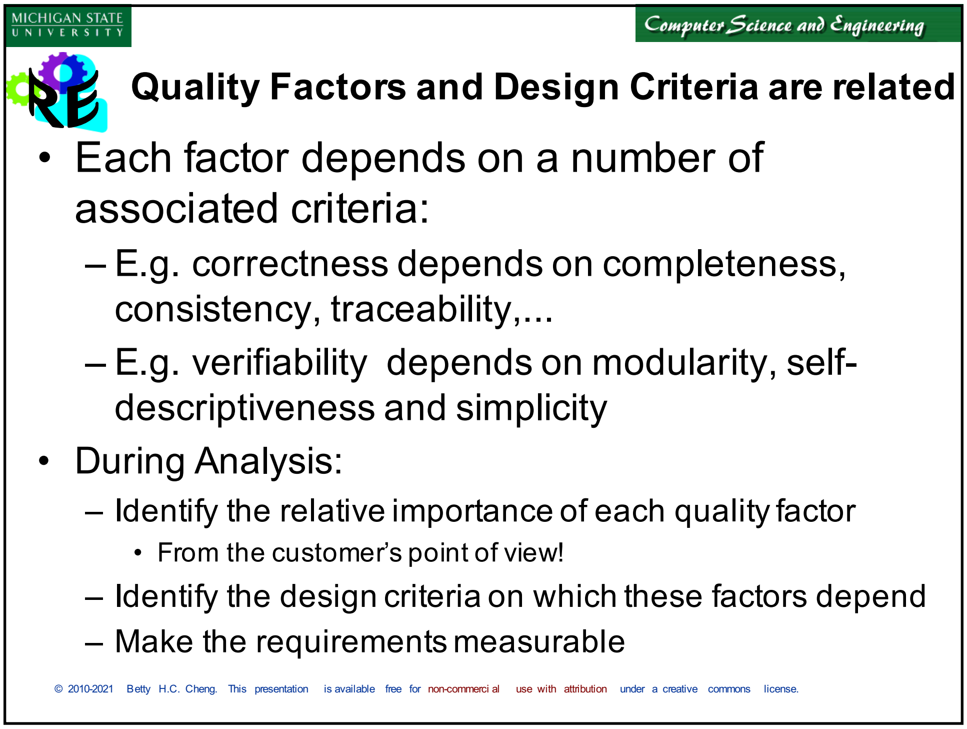
Fitness

- Quality is not a measure of software in isolation
  - it measures the relationship between software and its application domain
    - cannot measure this until you place the software into its environment…
    - …and the quality will be different in different environments!
  - during design, we need to *predict* how well the software will fit its purpose
    - we need good quality predictors (design analysis)
  - during requirements analysis, we need to *understand* how fitness-for-purpose will be measured
    - What is the intended purpose?
    - What quality factors will matter to the stakeholders?
    - How should those factors be operationalized?



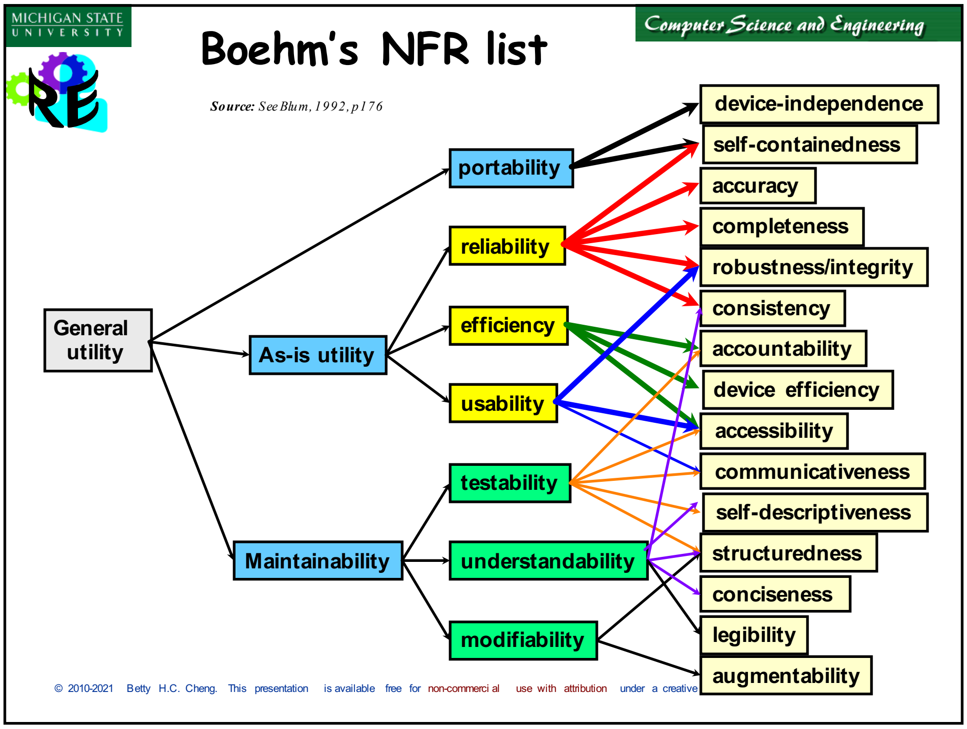
Factors vs. Criteria

- Quality Factors
  - These are customer-related concerns
    - Examples: efficiency, integrity, reliability, correctness, survivability, usability,...
- Design Criteria
  - These are technical (development-oriented) concerns such as anomaly management, completeness, consistency, traceability, visibility,...



Quality Factors and Design Criteria are related

- Each factor depends on a number of associated criteria:
  - E.g. correctness depends on completeness, consistency, traceability,...
  - E.g. verifiability depends on modularity, self-descriptiveness and simplicity
- During Analysis:
  - Identify the relative importance of each quality factor
    - From the customer's point of view!
  - Identify the design criteria on which these factors depend
  - Make the requirements measurable



Boehm's NFR list

Source: See Blum, 1992, p176

General utility

As-is utility

Maintainability

portability

reliability

efficiency

usability

testability

understandability

modifiability

device-independence

self-containedness

accuracy

completeness

robustness/integrity

consistency

accountability

device efficiency

accessibility

communicativeness

self-descriptiveness

structuredness

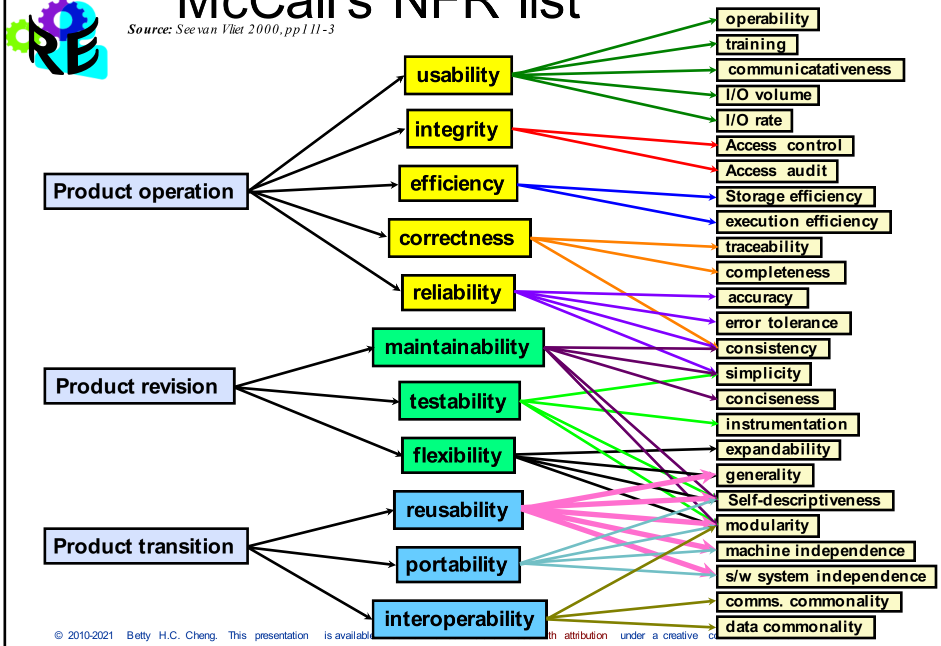
conciseness

legibility

augmentability

McCall's NFR list

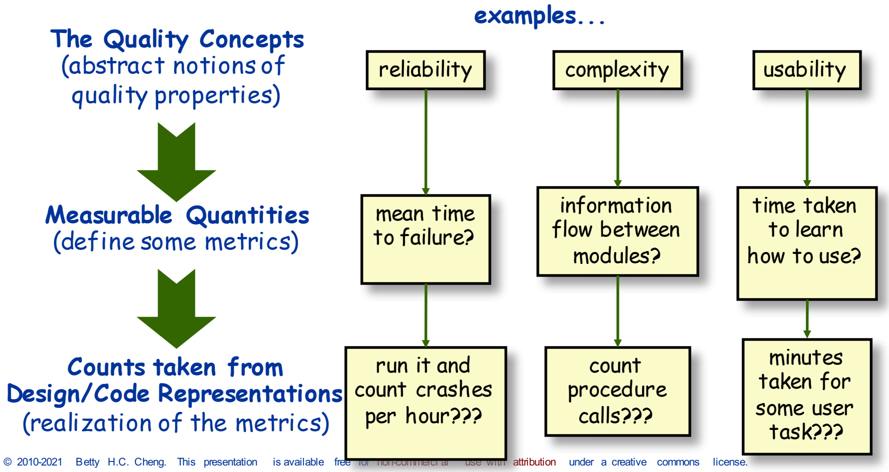
Source: *See van Vliet 2000, pp111-3*



Making Requirements Measurable

Source: *Budgen, 1994, pp60-1*

- We have to turn our vague ideas about quality into measurables

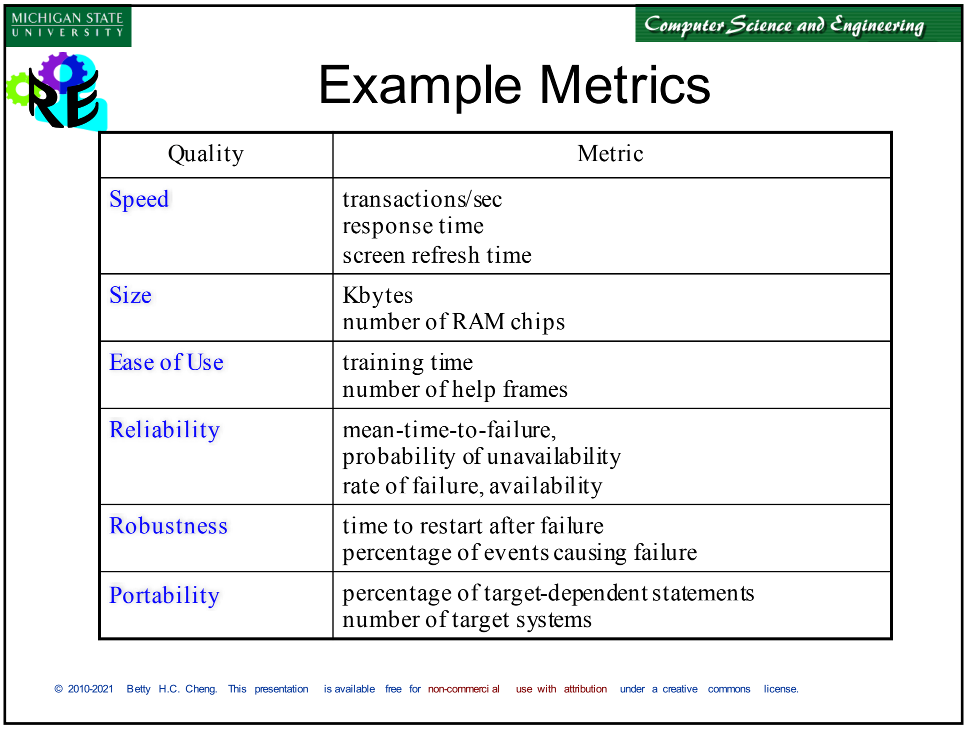

examples...

The Quality Concepts
(abstract notions of quality properties)

Measurable Quantities
(define some metrics)

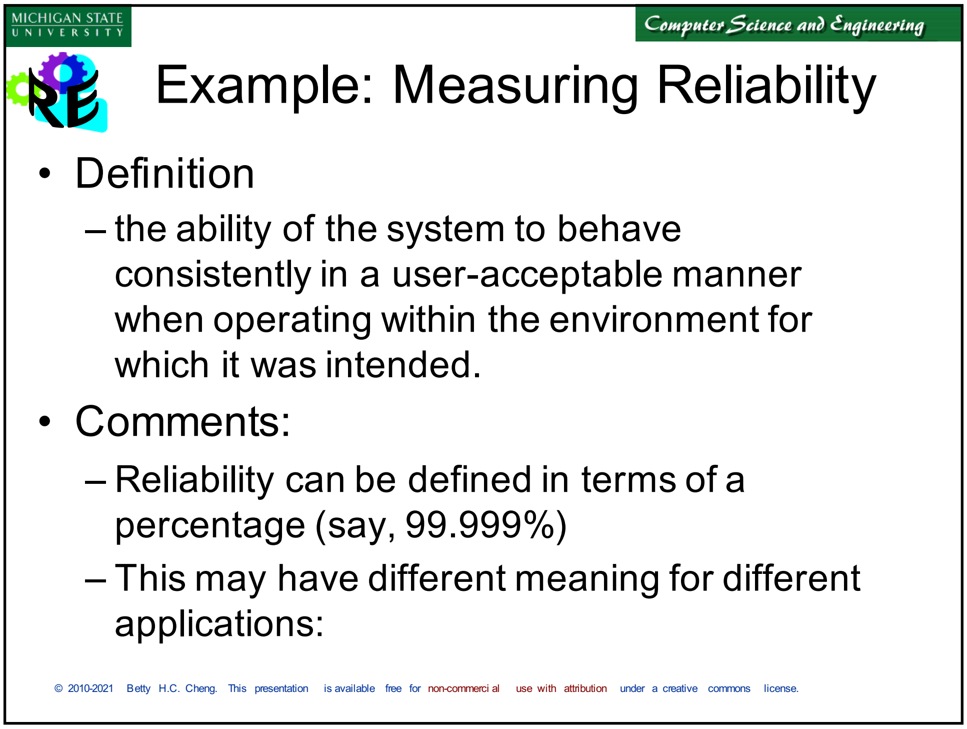
Counts taken from Design/Code Representations
(realization of the metrics)

reliability	complexity	usability
mean time to failure?	information flow between modules?	time taken to learn how to use?
run it and count crashes per hour???	count procedure calls???	minutes taken for some user task???



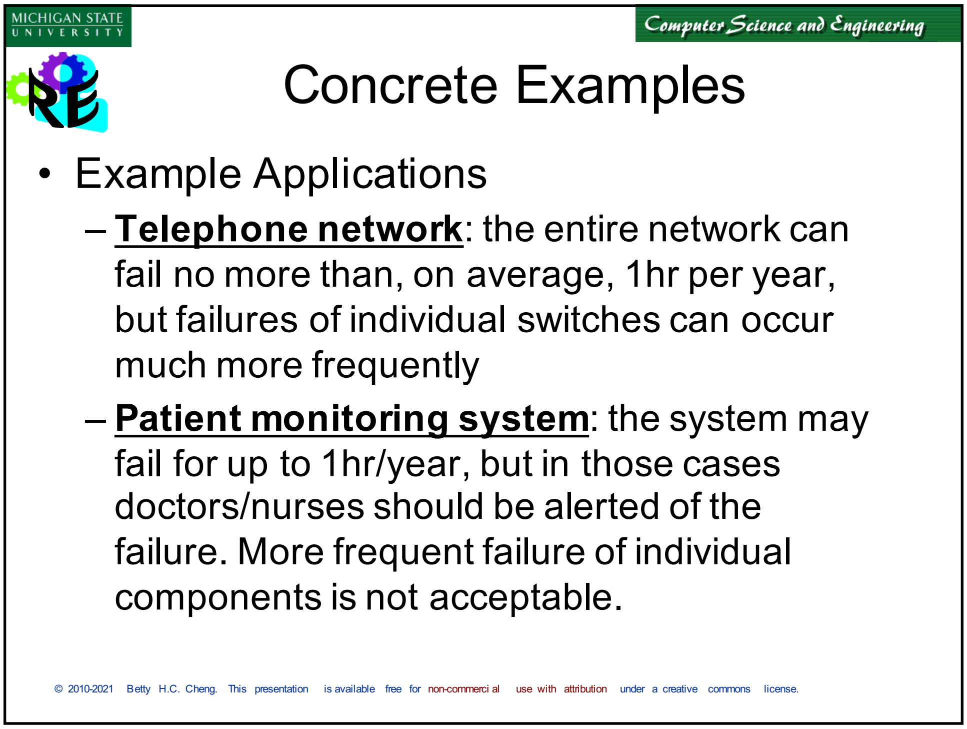
Example Metrics

Quality	Metric
Speed	transactions/sec response time screen refresh time
Size	Kbytes number of RAM chips
Ease of Use	training time number of help frames
Reliability	mean-time-to-failure, probability of unavailability rate of failure, availability
Robustness	time to restart after failure percentage of events causing failure
Portability	percentage of target-dependent statements number of target systems



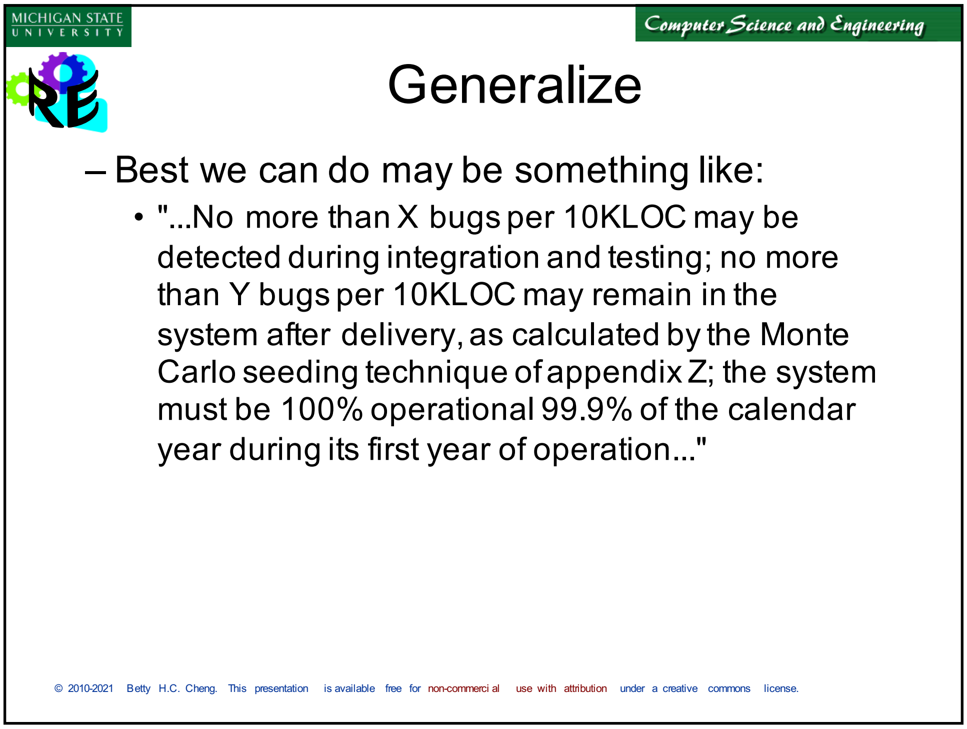
Example: Measuring Reliability

- Definition
  - the ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which it was intended.
- Comments:
  - Reliability can be defined in terms of a percentage (say, 99.999%)
  - This may have different meaning for different applications:



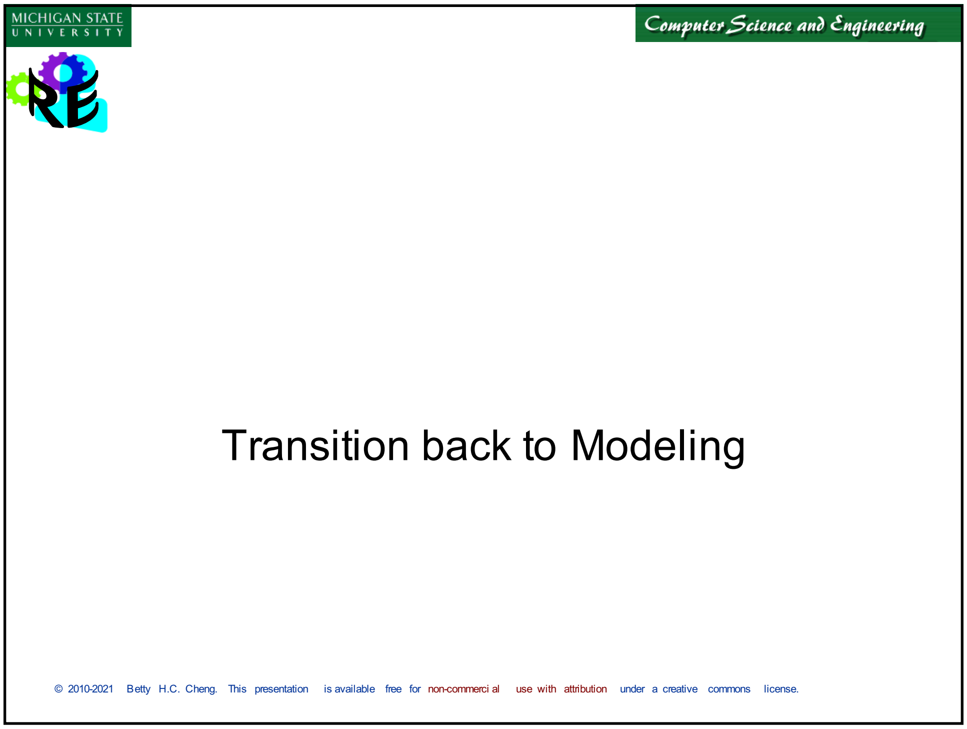
Concrete Examples

- Example Applications
  - **Telephone network**: the entire network can fail no more than, on average, 1hr per year, but failures of individual switches can occur much more frequently
  - **Patient monitoring system**: the system may fail for up to 1hr/year, but in those cases doctors/nurses should be alerted of the failure. More frequent failure of individual components is not acceptable.



Generalize

- Best we can do may be something like:
  - "...No more than X bugs per 10KLOC may be detected during integration and testing; no more than Y bugs per 10KLOC may remain in the system after delivery, as calculated by the Monte Carlo seeding technique of appendix Z; the system must be 100% operational 99.9% of the calendar year during its first year of operation..."



Transition back to Modeling