EEE499 - Real-Time Embedded System Design

Software Fault Tolerance

Failures

- when the behavior of a system deviates from that of its specification.
- normally associated with the notion that something which once functioned properly, no longer does so.

- in this <u>classical</u> sense, software does not truly fail; if the software does the wrong thing, it will always do the wrong thing under identical circumstances

Failures & Specifications

- traditionally (software) requirements are viewed as a positive specification of a system
- an attempt to define the failures of a system can be viewed as a negative specification of a system
- while both are likely essential, the latter is rarely formally identified in projects
  - system engineers will often need the failure definitions

Faults

- an unsatisfactory condition or state
  - actions, timing, sequence or amount
- random faults - in the context of above, failures are random faults which occur when a component breaks in the field. Random faults only occur in physical entities.
- systematic faults - are intrinsic in the design or implementation of a component. Errors or software defects (bugs) are systematic faults.

Faults (cont’d)

- hardware - faults can be random or systematic
  - traditional bath-tub curve phenomenon
- software - faults are always systematic
  - software neither breaks nor wears out
  - all software contains latent defects (bugs)
  - estimates:
    - 20,000 defects per Million LOC
    - 90% found in testing
    - 200 found in early life
    - 1800 latent defects (0.18% defect rate)

Failure Modes

- the classification of a system’s failures according to the impact they have on the services it delivers.
- software failure modes:
  - value domain
  - timing domain - early, omission, late
  - arbitrary (combination of value and timing)

Failure Types

- single point failures - the failure of a system due to a single component failure.
  - h/w: car brake master cylinder
  - s/w:?
- common cause failures - the failure of more than one component due to a common event or cause.
  - h/w: power supply failure
  - s/w:?

Fault Prevention

- fault avoidance - limit the introduction of faulty components during construction
  - rigorous (formal) specifications
  - proven design methodologies
  - proper selection of programming language
  - engineering / development environments (IDEs)
- fault removal - finding and removing causes of error
  - reviews, inspections, testing, verification, validation
- but what if faults can not be prevented?

Fault Tolerance

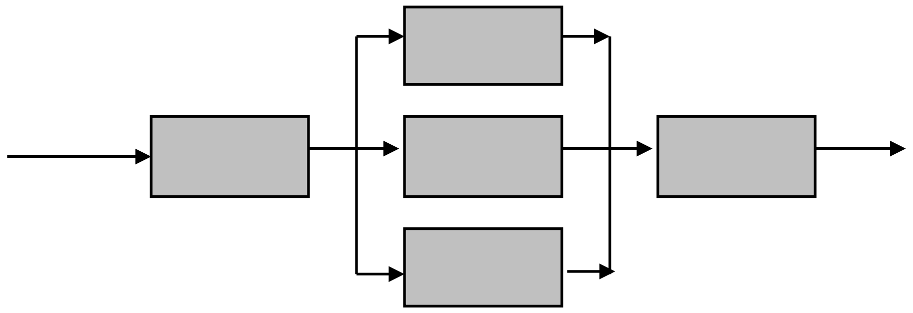
- the ability of a system to continue functioning even in the presence of faults.
  - full fault tolerance - system continues to operate with no significant loss of functionality (for a limited time)
  - graceful degradation - system continues to operate but in a degraded mode
  - fail safe - system transitions to a safe state prior to shut-down or as a result of a particular fault

Redundancy

- in order to achieve fault tolerance in software some degree of redundancy is required.
  - *{In this sense, the definition of redundancy extends to include additional code that would not be required for normal operation}*
- goal - minimize redundancy, maximize R(t)
- paradox - too much may decrease R(t)

Static Redundancy

- redundant components are used to <u>mask</u> errors
- h/w: N Modular Redundancy (NMR)



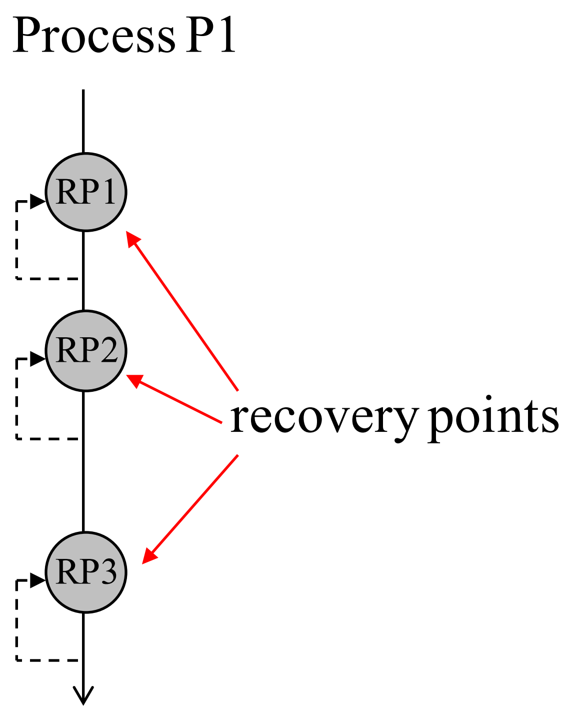
- s/w: N-version programming
  - N redundant versions of the same module under a common controller and a voting scheme

Static Redundancy

- assumptions
  - complete, consistent, unambiguous specification
  - independent failure modes
    - separate development teams, processors, languages, fault-tolerant communication lines, …
- issues (problems)
  - specifications are a major contributor to defects
  - independence is often not a reasonable assumption
  - $$$

Dynamic Redundancy

- redundant components used to <u>detect</u> errors
- h/w: data parity bit checks
- s/w: recovery blocks
- phases
  - error detection
  - damage assessment/control
  - error recovery
  - fault treatment/continue



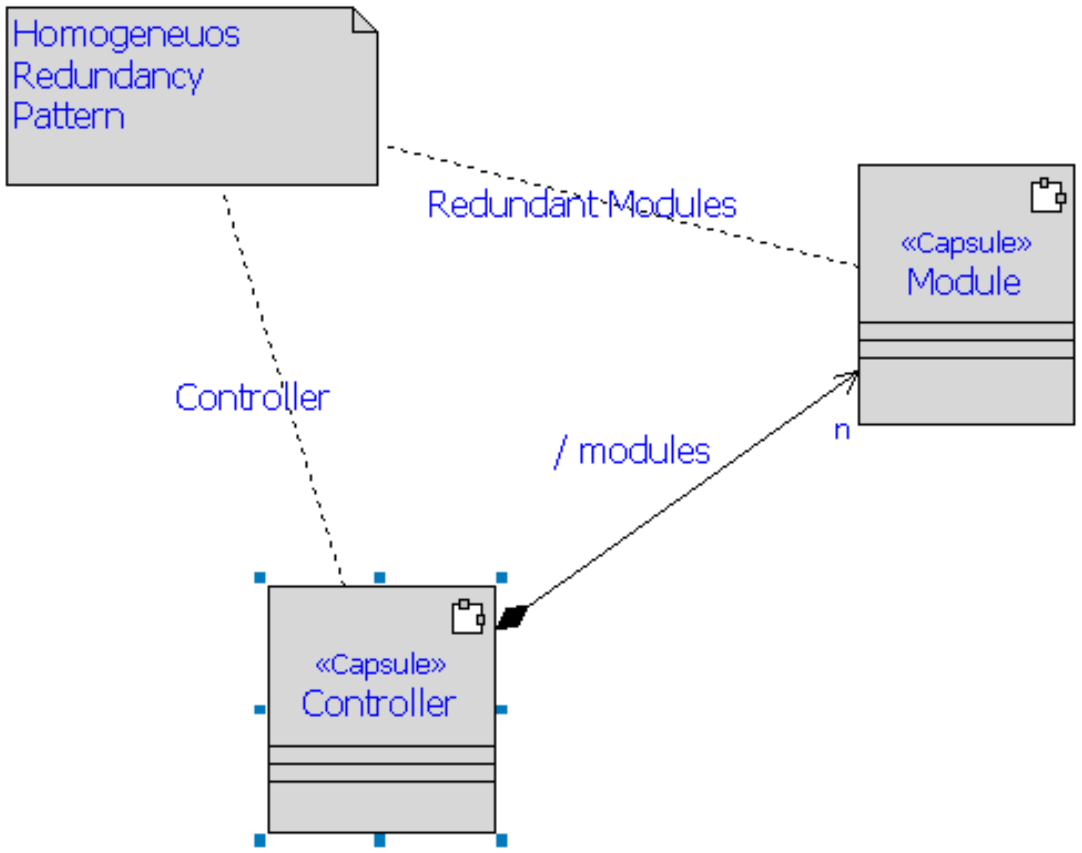
Dynamic Redundancy

- assumptions
  - alternate modules only execute based upon error detection
  - still largely dependent upon the specification
  - known and unknown failure modes <u>may</u> be handled
- issues (problems)
  - backward error recovery can’t always undo damage
  - $$

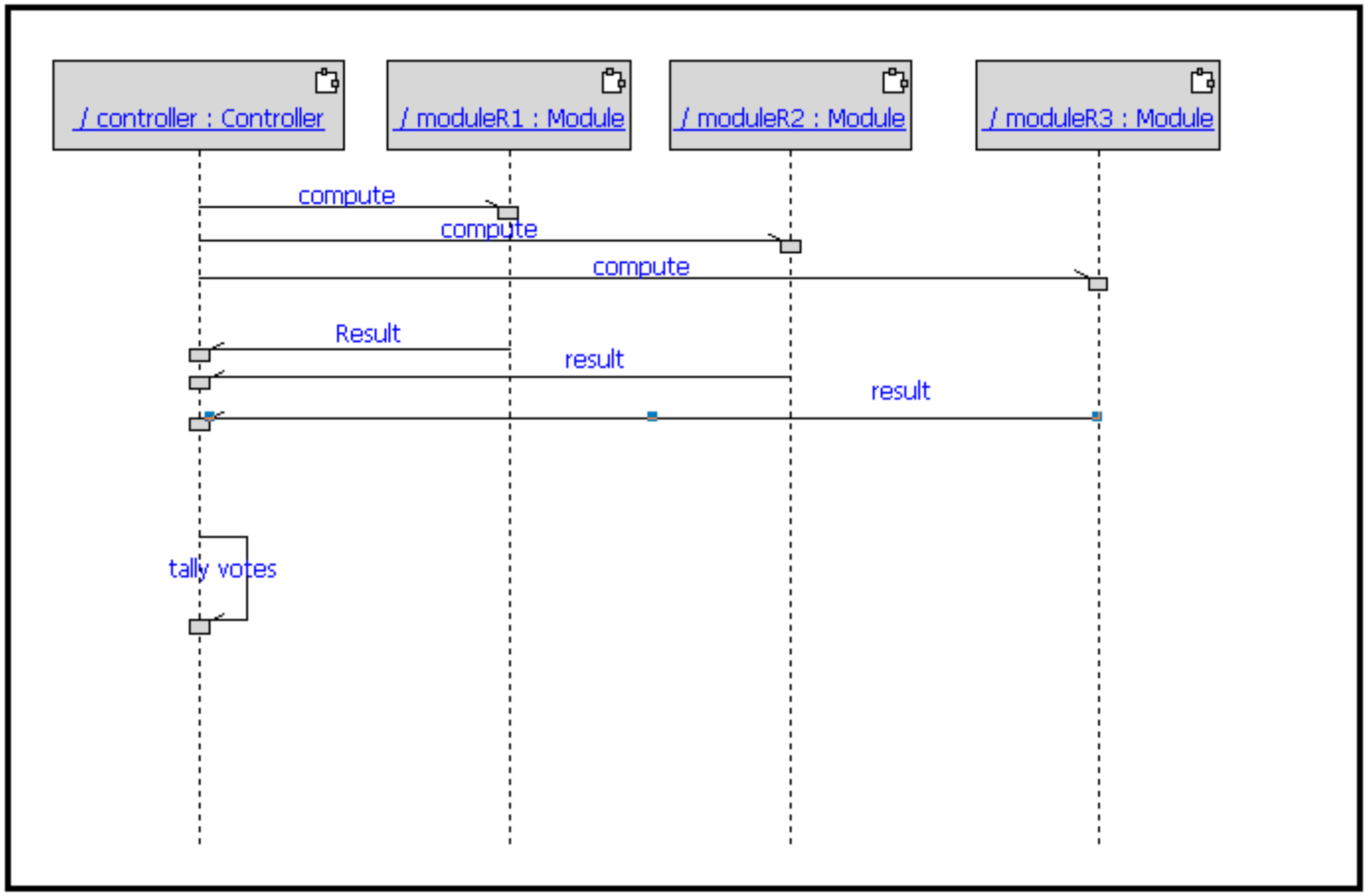
A Few Fault Tolerance Patterns

- Homogeneous Redundancy Pattern
  - protects against hardware (random) failures only
- Diverse Redundancy Pattern
  - protects against systematic & random faults
- Monitor Actuator Pattern
  - specialization of diverse redundancy
- Watchdog Pattern

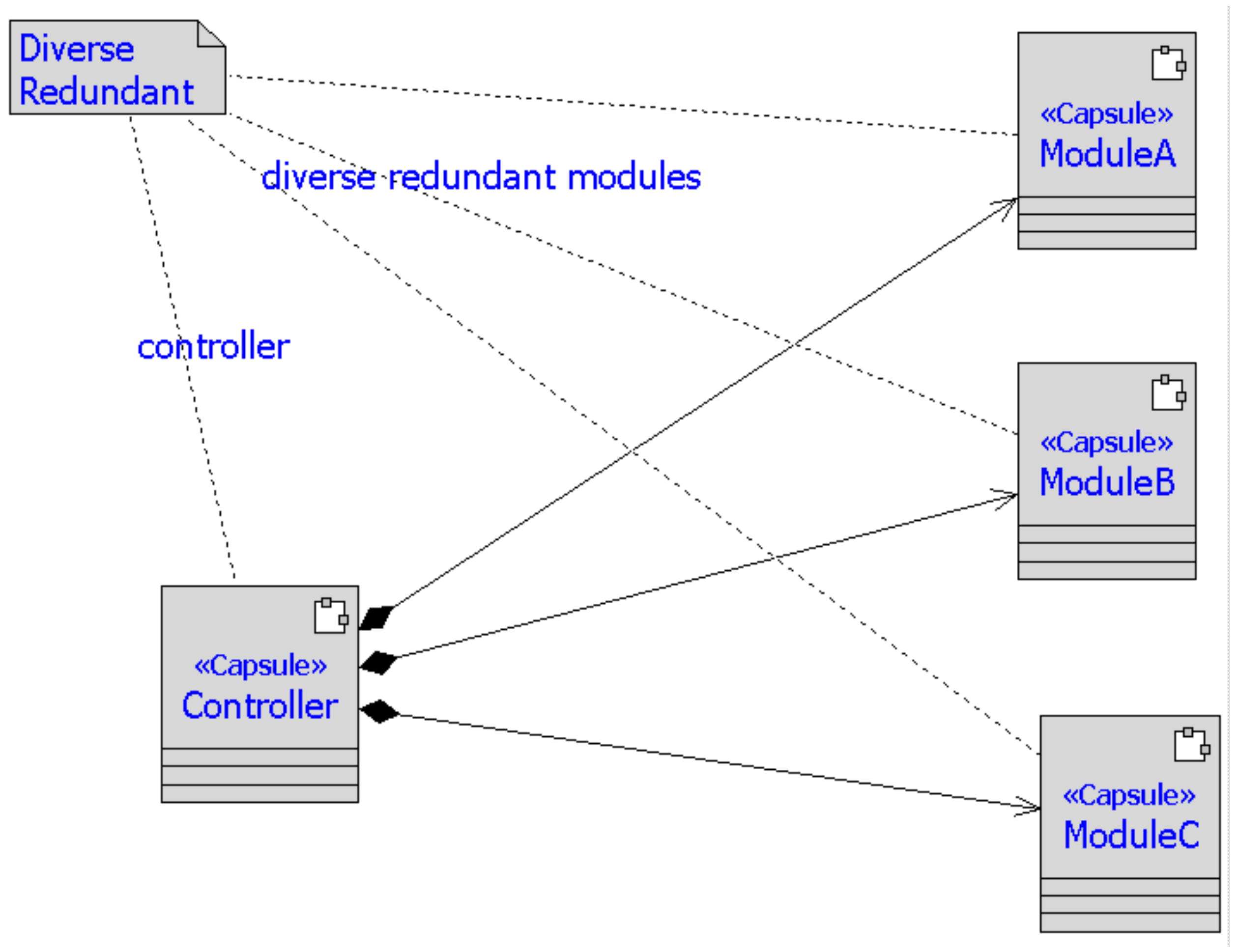
Homogeneous Redundancy Pattern



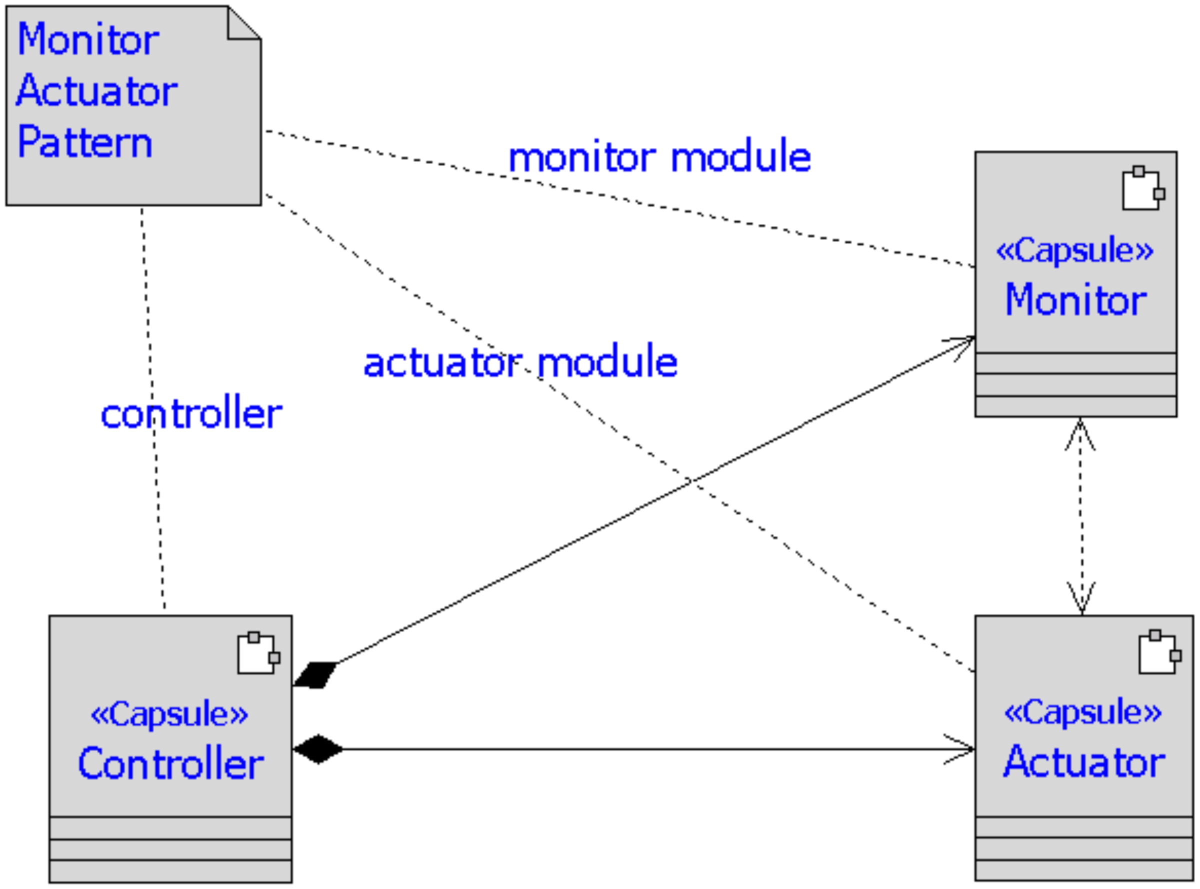
Homogeneous Redundancy Pattern



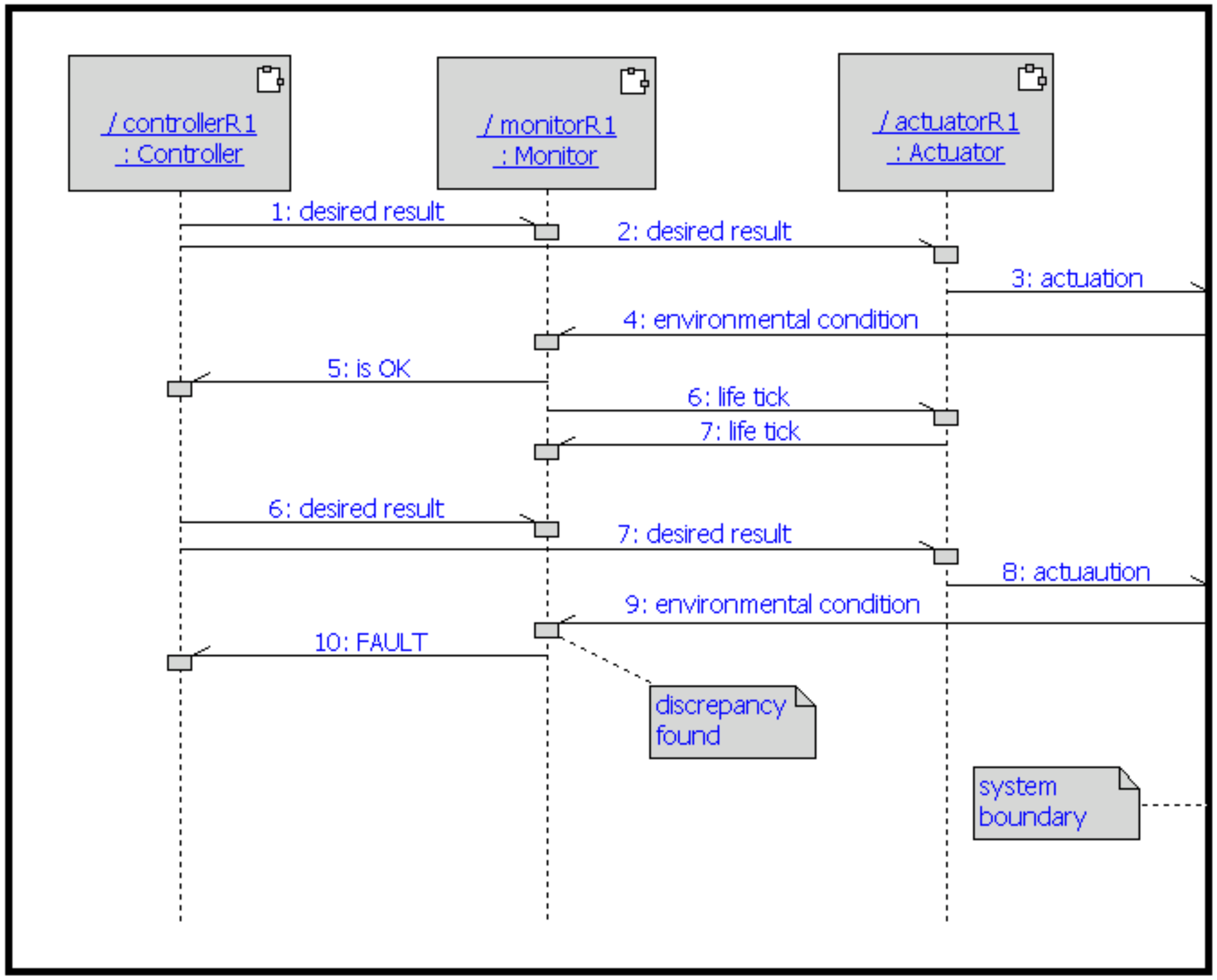
Diverse Redundancy Pattern



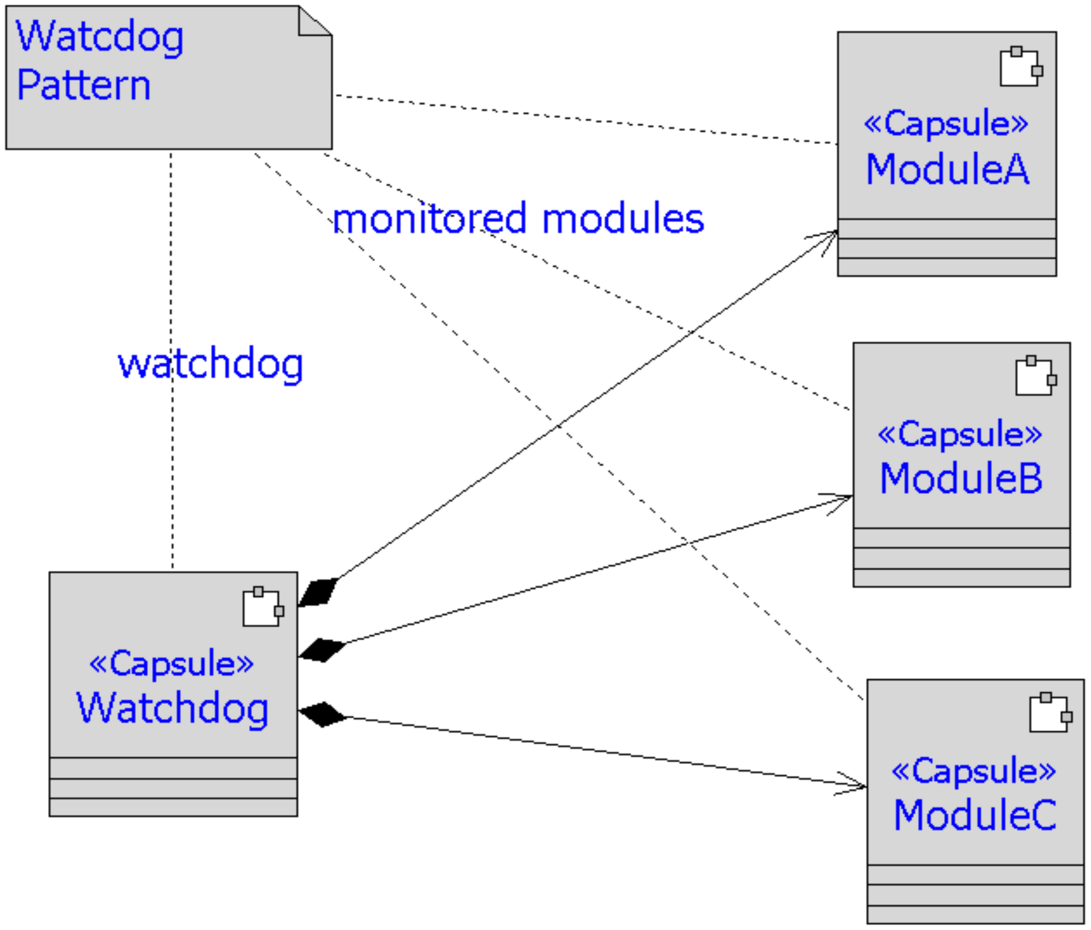
Monitor-Actuator Pattern



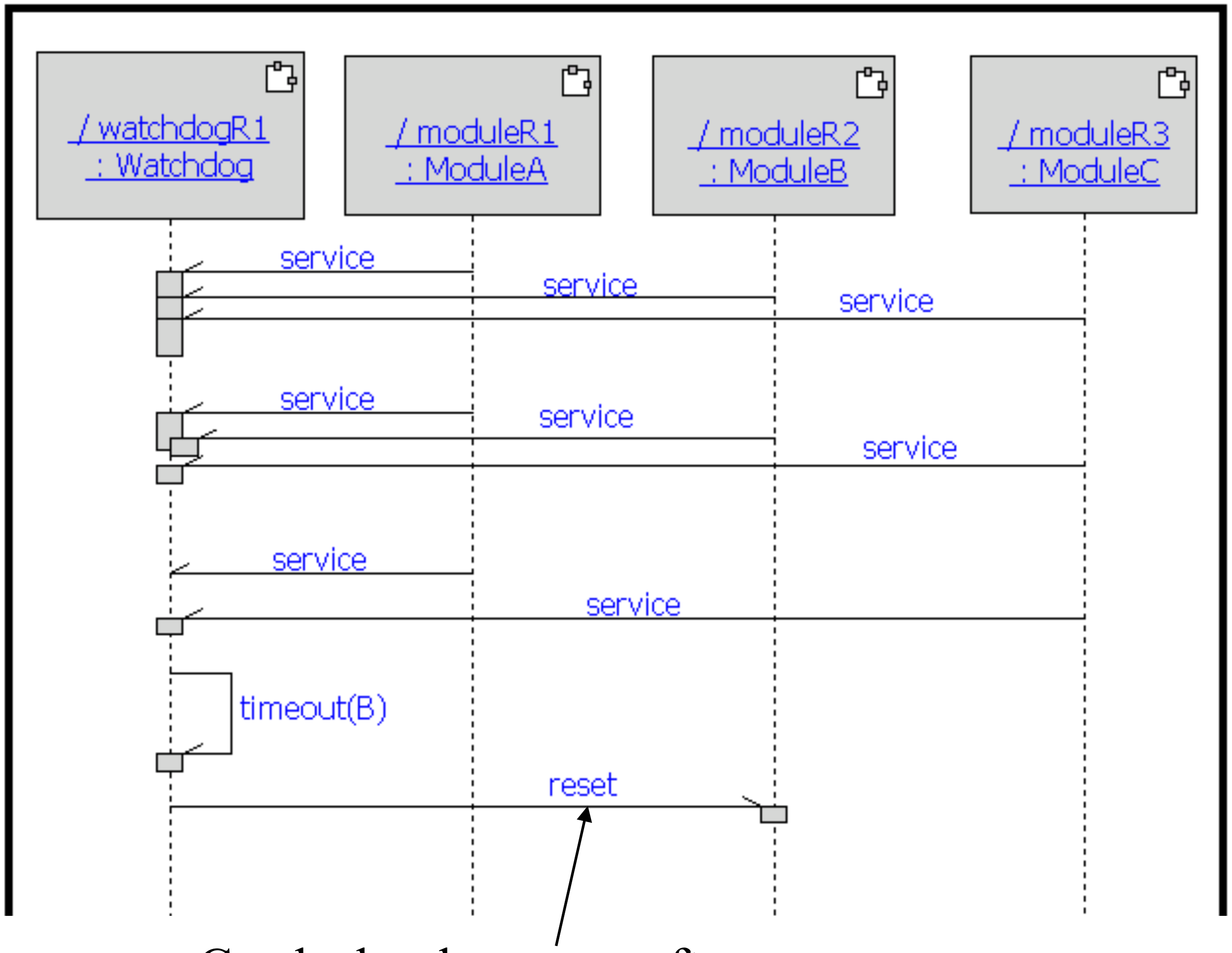
Monitor-Actuator Pattern



Watchdog Pattern



Watchdog Pattern



Can be hardware or software

Reliability

- reliability, R(t) - the probability that, when operating under stated environmental conditions, a system will perform its intended function adequately for a specified interval of time.
- a measure of the success with which a system conforms to some authoritative specification of its behavior
- most frequent hardware metric - MTBF
  - failure rate is more universal in software

Reliability & Time

- as stated in its definition, reliability is evaluated on a time interval
  - R(t = 6 hours) = 95%

- in terms of software reliability, we must distinguish three forms of time:
  - calendar time - standard elapsed time
  - clock time - the time a processor is actually running during any calendar time
  - execution time- the time any software program is executing on the given processor

Reliability & Environment

- again, as stated in its definition, reliability is a function of the operational environment
  - more commonly used is a system's operational profile
- an operational profile of a piece of software is a representation of the various input states and their respective probabilities
  - often these profiles can be further divided along modes of operation
  - for example an application has a different profile during start-up mode than in full operation
- test cases / test data is selected accordingly

Reliability Definition

- reliability, R(t) - the probability that, when operating under stated environmental conditions, a system will perform its intended function for a specified interval of time.

- reliability is evaluated on a time interval
  - R(t = 6 hours) = 95%

- thus, reliability is a function of some failure rate (intensity) and a desired time interval

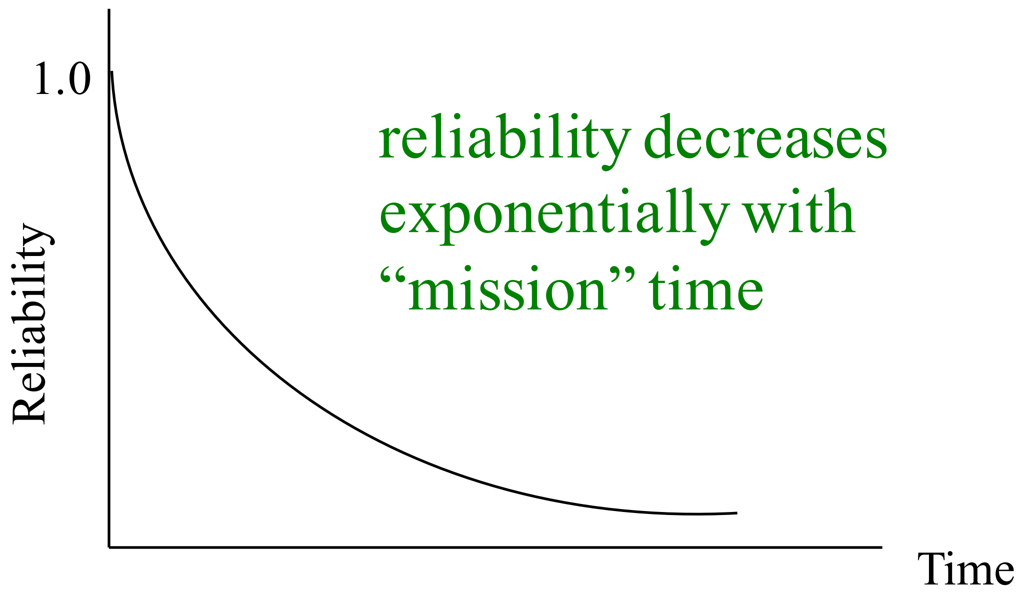
Uses of Reliability

- a measure of development status
  - given a reliability goal (a set failure rate), you may determine remaining schedule, costs, tests, etc prior to release of the software
- to monitor operational performance
  - is the reliability such that major maintenance is justified
- provides insight into the software product (quality) and the development process

Quantifying Component Reliability

- Given components from a Homogeneous Poisson Process with a failure intensity (λ), then reliability can be expressed as:

$$R(T) = e^{-\lambda T}$$



Component Reliability - Example 1

- Assume that a capacitor has been tested and determined to have a mean-time-between-failure (MTBF) of 200 hours,
  - what is the probability that the capacitor will not fail during 8 hours of (normal) use in the lab?

Solution:

$\lambda = 1/\text{MTBF} = 0.005$ failures per hour

$R(T=8 \text{ hours}) = e^{-(0.005)(8)}$

$= 0.961$ or 96.1%

Component Reliability - Example 2

- Given an unmanned space probe with a requirement to operate failure free for a 25 year mission with a probability of 95%,
  - what is the required system MTBF?

Solution:

$$\lambda = -\ln(R(T)) / T$$

$$= -\ln(0.95) / [(25)(365)(24)]$$

$$= 0.0000002 \text{ failures per hour}$$

MTBF = 5,000,000 hours!

Software Failure Intensity

- As most software reliability values are stated in terms of execution time, we need to be able to convert these values to clock time:
  - if we define:
    - execution time as λ_τ
    - clock time as $\lambda_{t,}$
  - then, $\lambda_t = \rho_c \lambda_\tau$
  - where, ρ_c represents average utilization

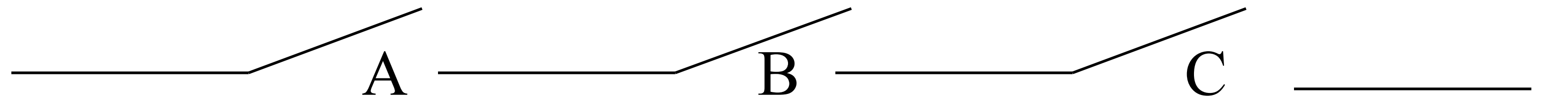
Software Failure Intensity - Example

- Given a periodic task with an estimated one hour (execution time) reliability of 99.5%. In addition, we know the task runs every 200 msec with an average computation time of 2000 μsec.
  - determine the clock-time-based failure intensity

$\lambda_\tau = -\ln(R(\tau=1))/\tau = -\ln(0.995)/1 = 0.005$ f/hr

$\lambda_t = \rho_c\ \lambda_\tau = (0.01)\ (0.005) = 0.00005$ f/hr

System Reliability - Serial Systems

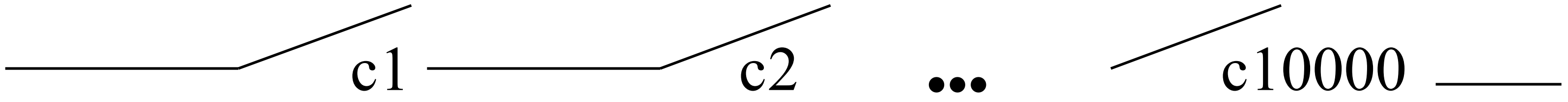


- $R_{sys} = \Pi R_i$ for all i components

- Example
  - Given $R_A = 90\%$, $R_B = 97.5\%$, $R_C = 99.25\%$
  - $R_{sys} = (.9)(.975)(.9925) = 87.1\ \%$

*Assumes that all components are reliability-wise independent.

Serial Systems - Example



- Given a space probe with 10,000 identical components and a 25 year 95% reliability requirement:
  - what is the required component failure rate?

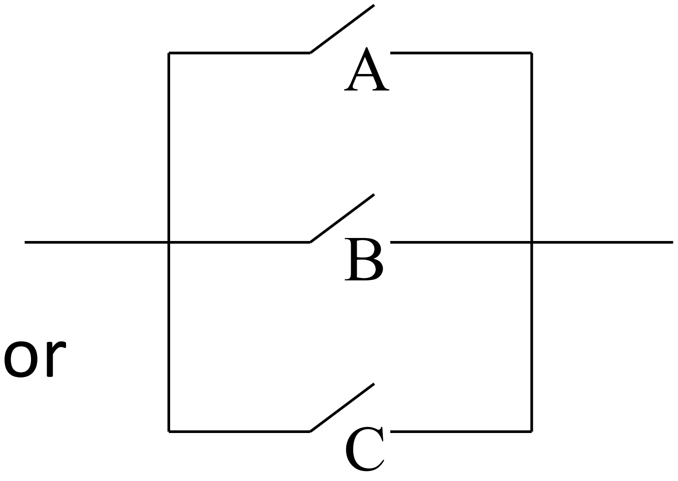
Solution:

$$R_C = (R_{sys})^{1/10000} = 0.999995$$

$$\lambda_c = -\ln(.999995)/[(25)(365)(24)] = 2.28\text{ E-11}$$

System Reliability - Parallel Systems

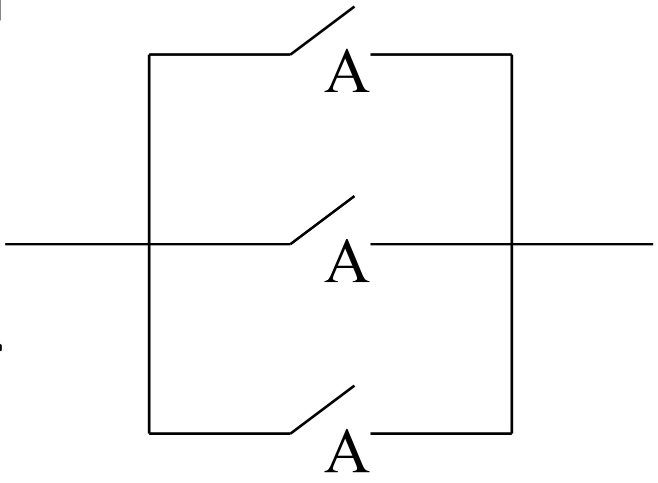
- Define the probability of failure as Qi = 1- R_i, then
- $Q_{sys} = \Pi Q_i$
  - it follows that $R_{sys} = 1 – Q_{sys}$, or
- $R_{sys} = 1 - \Pi (1 - R_i)$



- Example
  - Given R_A = 90%, R_B = 97.5%, R_C = 99.25%
  - R_{sys} = 1 - (.1)(.025)(.0075) = 99.998%

System Reliability - k out of n

- a special case of the parallel system configuration wherein it is required that only k out of n identical components are needed for success

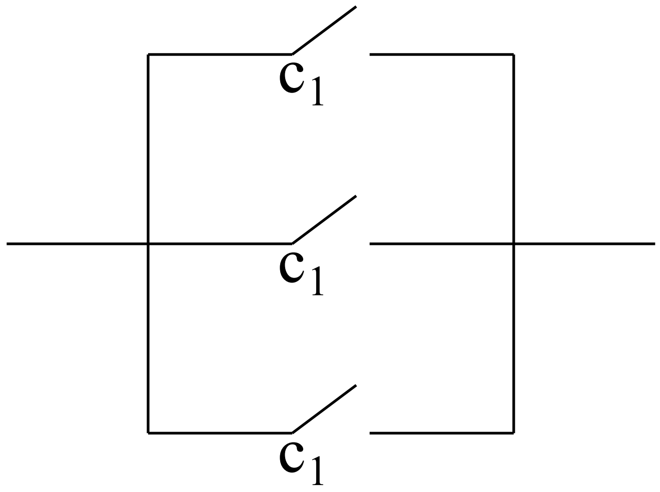


- $R_{sys} = \sum_{i=k}^{n} [\binom{n}{i}] R_c^{i} (1 - R_c)^{n-i}$

▪ where "n choose k", $[\binom{n}{k}] = n! / (k!(n-k)!)$

System Reliability - k out of n

- Example
  - assume that the reliability of c_1 is 90% and you need at least two for the system to function



solution:

$$R_{sys} = \sum_{i=2}^{3} \left[{3 \atop i} \right] R_c^{\,i} (1 - R_c)^{3-i}$$

$$= \left[{3 \atop 2} \right] R_c^{\,2} (1 - R_c) + \left[{3 \atop 3} \right] R_c^{\,3} \overset{1}{\cancel{(1 - R_c)^0}}$$

$$= (3)(.81)(.1) + (1)(.729)(1)$$

$$= 97.2\%$$

Exercise - System Components

- Consider an EW system made up of the following components:
  - 2 touch displays, only one of which need function for the system to function
  - each touch display contains identical software (meaning that you should treat the s/w as a stand-alone module)
  - one system processor and the system integration software
  - an ECM with embedded jammer software
- Assume that all hardware utilization is 100%
  - in other words all hardware functions for the entire mission duration

Exercise - Component Data

Component	Failure Rate (execution hours)	Utilization	R(t=4)
TD	-	-	.95
TD s/w	0.002	0.65	?
SP	-	-	.98
SP s/w	.010	.95	?
ECM	-	-	.925
ECM s/w	.012	.35	?

Exercise - Questions

- a) Determine the overall system reliability for a 4 hour mission?
- b) What is the probability of having at least one functional display for a 2 hour mission?
- c) What is the weakest link in the system? What could be done to improve the overall system reliability (assuming that you can not significantly change the component reliabilities without serious redesign). Draw the new reliability block diagram and find R_{sys}

Exercise - Questions (continued)

- d) Assume that we now have 4 equivalent ECM subsystems (combined hardware and software) and that for the system to be functional any 3 of the 4 must be functional.
- Draw the new reliability block diagram and calculate the system reliability.

References

[1] Burns and Wellings, “Real-Time Systems and Programming Languages”, Chap 5.

[2] Douglass, “Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns”, Chap 3.

[3] Musa, J.D., Iannino, A., Okumoto,K. “Software Reliability - Measurement, Prediction, Application”, Chapter 4, McGraw-Hill 1987.