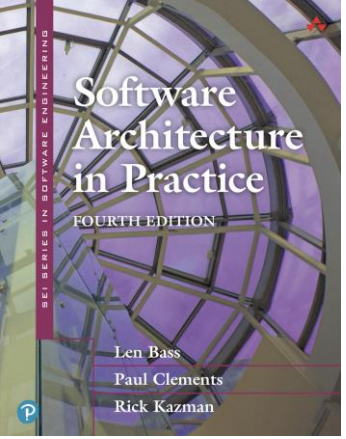
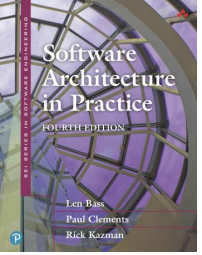


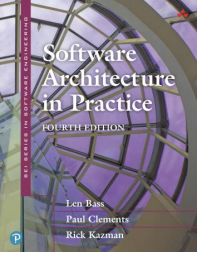
Chapter 10: Safety

Giles: Well, for god's sake, be careful. . . . If you should be hurt or killed, shall take it amiss.
Willow: Well, we try not to get killed. That's part of our whole mission statement: Don't get killed.
Giles: Good.
—Buffy the Vampire Slayer, Season 3



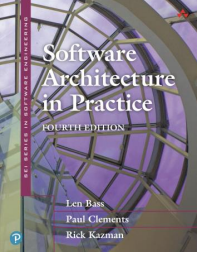
Chapter Outline

- What is Safety?
- Safety General Scenario
- Tactics for Safety
- Tactics-Based Questionnaire for Safety
- Patterns for Safety
- Summary



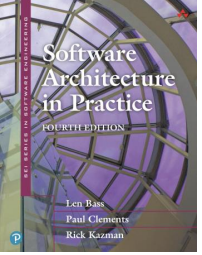
What is Safety?

- "Don't kill anyone" should be a part of every software architect's mission statement.
- Safety is concerned with a system's ability to avoid straying into states that lead to damage, injury, or loss of life. These unsafe states can be caused by a variety of factors:
  - *Omissions*
  - *Commission*
  - *Timing*
  - *Problems with system values*
  - *Sequence omission and commission*
  - *Out of sequence*
- Safety is concerned with detecting and recovering from these unsafe states to prevent or minimize harm.



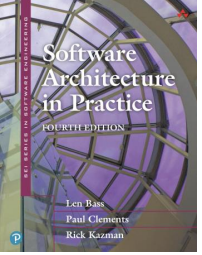
What is Safety?

- The unsafe state should be recognized and the system should be made safe through
  - Continuing operations
  - Shutting down (fail safe), or
  - Transitioning to a state requiring manual operation
- In addition, the unsafe state should be reported and/or logged.
- Architecting for safety begins by identifying safety-critical functions using techniques such as failure mode and effects analysis (FMEA; also called hazard analysis) and fault tree analysis (FTA).



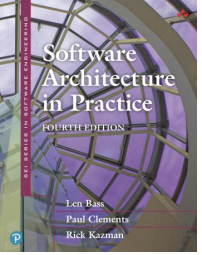
Safety General Scenario

Portion of Scenario	Description	Possible Values
Source	A data source (a sensor, a software component that calculates a value, a communication channel), a time source (clock), or a user action	Specific instances of a: ▪ Sensor ▪ Software component ▪ Communication channel ▪ Device (such as a clock)
Stimulus	An omission, commission, or occurrence of incorrect data or timing	A specific instance of an omission: ▪ A value never arrives. ▪ A function is never performed. A specific instance of a commission: ▪ A function is performed incorrectly. ▪ A device produces a spurious event. ▪ A device produces incorrect data. A specific instance of incorrect data: ▪ A sensor reports incorrect data. ▪ A software component produces incorrect results. A timing failure: ▪ Data arrives too late or too early. ▪ A generated event occurs too late or too early or at the wrong rate. ▪ Events occur in the wrong order.
Environment	System operating mode	▪ Normal operation ▪ Degraded operation ▪ Manual operation ▪ Recovery mode



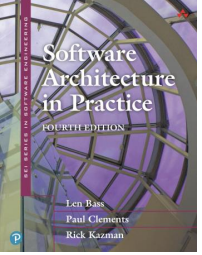
Safety General Scenario

Artifacts	The artifact is some part of the system.	Safety-critical portions of the system
Response	The system does not leave a safe state space, or the system returns to a safe state space, or the system continues to operate in a degraded mode to prevent (further) injury or damage or to minimize injury or damage. Users are advised of the unsafe state or the prevention of entry into the unsafe state. The event is logged.	Recognize the unsafe state and one or more of the following: ▪ Avoid the unsafe state ▪ Recover ▪ Continue in degraded or safe mode ▪ Shut down ▪ Switch to manual operation ▪ Switch to a backup system ▪ Notify appropriate entities (people or systems) ▪ Log the unsafe state (and the response to it)
Response measure	Time to return to safe state space; damage or injury caused	One or more of the following: ▪ Amount or percentage of entries into unsafe states that are avoided ▪ Amount or percentages of unsafe states from which the system can (automatically) recover ▪ Change in risk exposure: size(loss) * prob(loss) ▪ Percentage of time the system can recover ▪ Amount of time the system is in a degraded or safe mode ▪ Amount or percentage of time the system is shut down ▪ Elapsed time to enter and recover (from manual operation, from a safe or degraded mode)

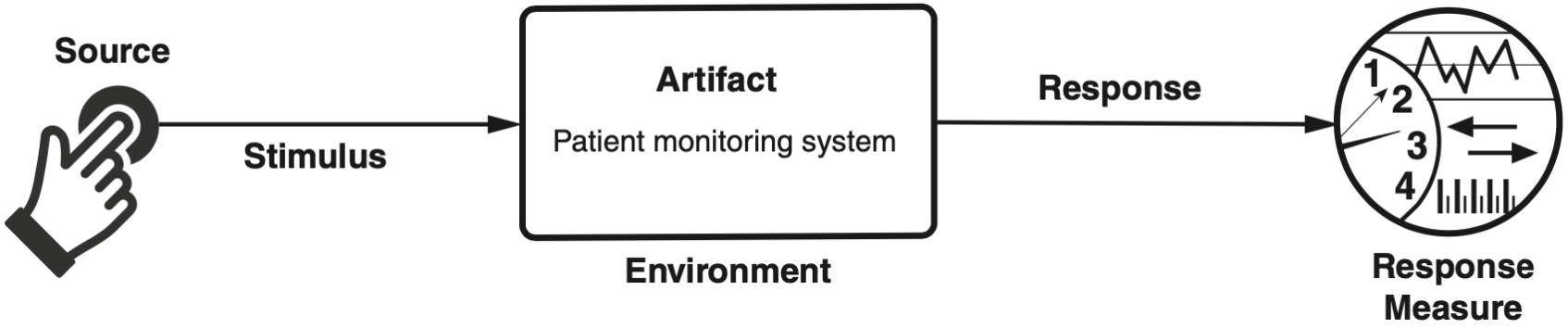


Sample Concrete Safety Scenario

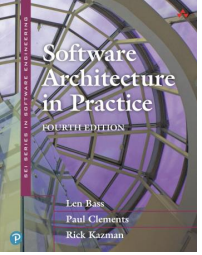
- *A sensor in the patient monitoring system fails to report a life-critical value after 100 ms. The failure is logged, a warning light is illuminated on the console, and a backup (lower-fidelity) sensor is engaged. The system monitors the patient using the backup sensor after no more than 300 ms.*



Sample Concrete Safety Scenario

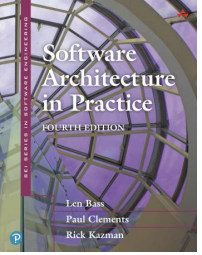


A sensor	Fails to report a life-critical value after 100 ms	Normal operations	The failure is logged, a warning light is illuminated on the console, and a backup (lower fidelity) sensor is engaged; the system monitors the patient using the backup sensor	After no more than 300 ms

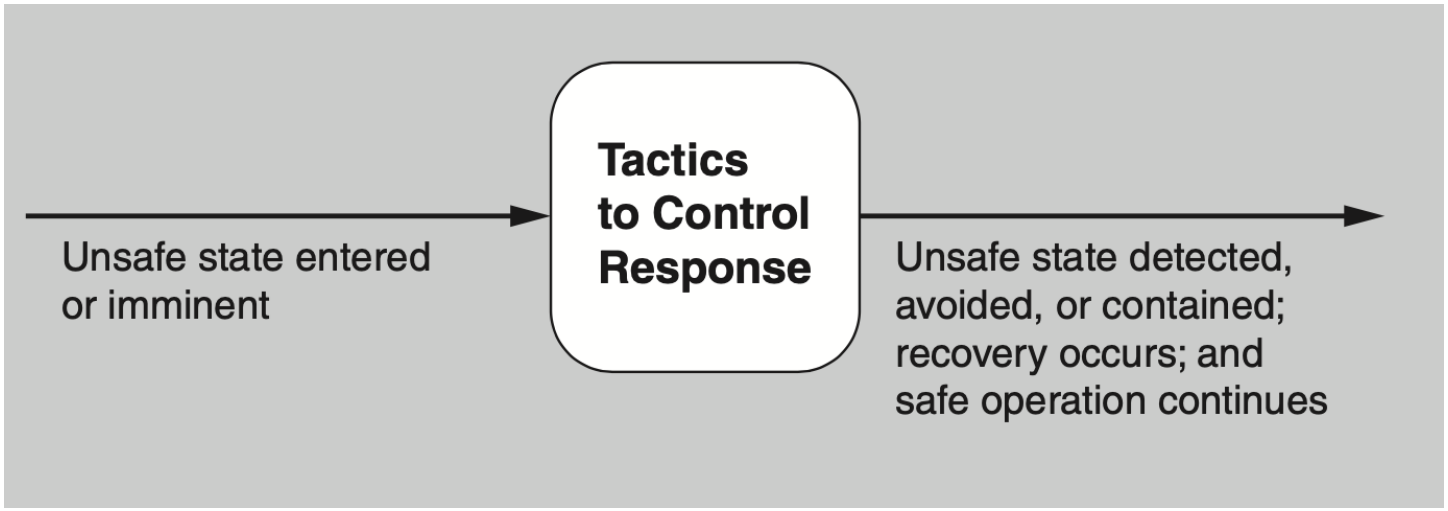


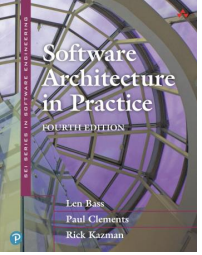
Goal of Safety Tactics

- Safety tactics may be broadly categorized as unsafe state avoidance, unsafe state detection, or unsafe state remediation.

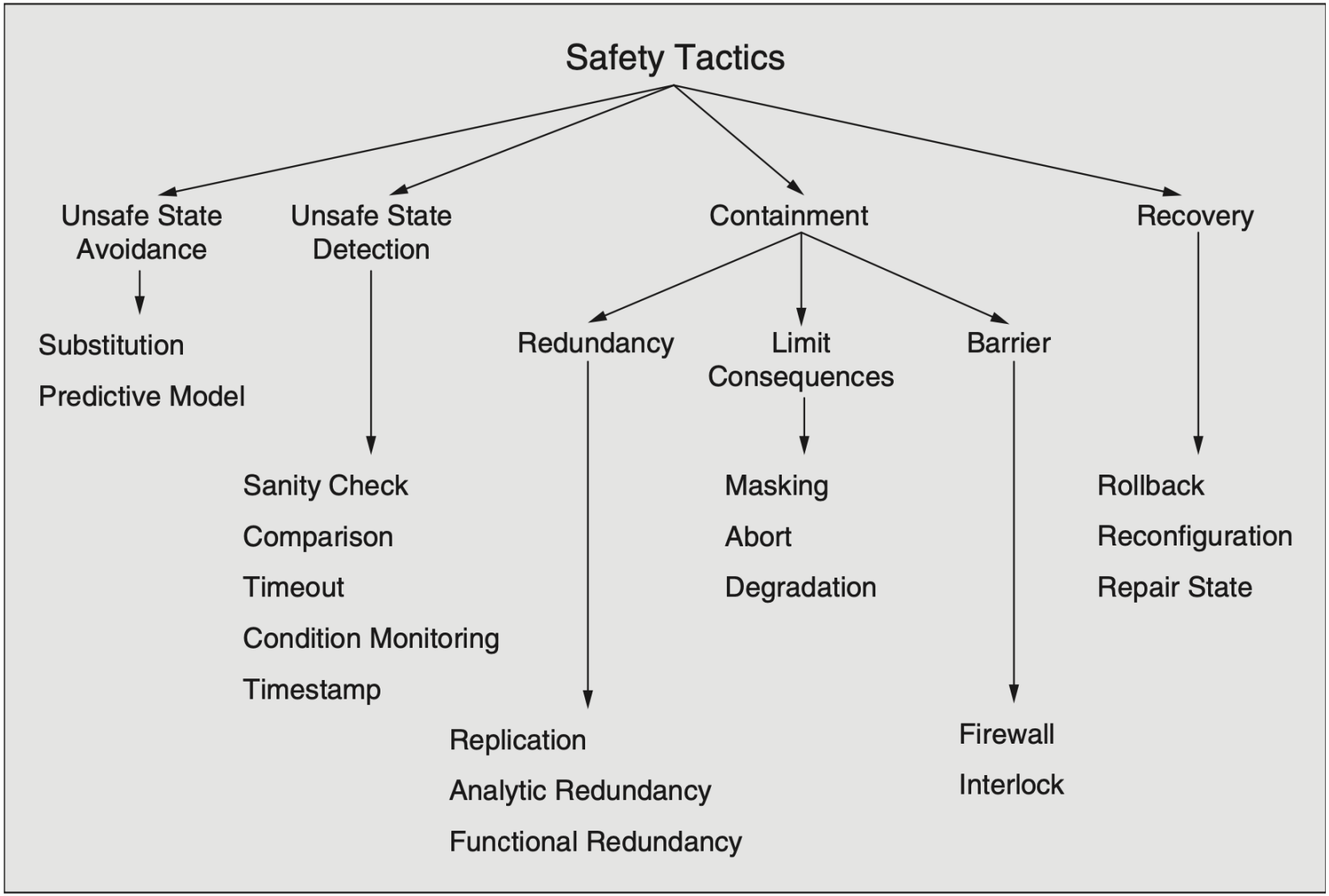


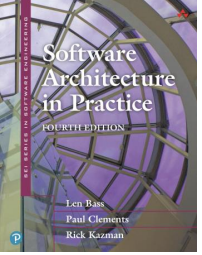
Goal of Safety Tactics





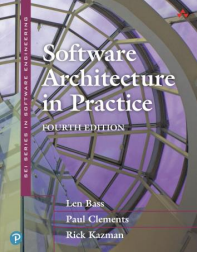
Safety Tactics





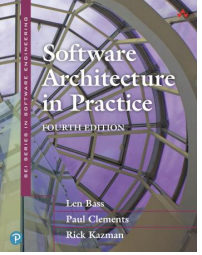
Unsafe State Avoidance

- *Substitution.* This tactic employs protection mechanisms—often hardware-based—for potentially dangerous software features. For example, devices such as watchdogs, monitors, and interlocks can be used in lieu of software versions. Software versions of these mechanisms can be starved of resources, whereas a separate hardware device provides and controls its own resources.
- *Predictive Model*. The predictive model tactic (introduced in Chapter 4), predicts the state of health of system processes/resources to provide early warning of potential problems.



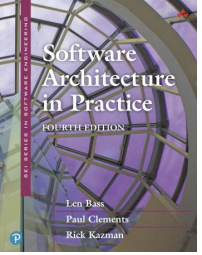
Unsafe State Detection

- *Timeout.* The timeout tactic is used to determine whether the operation of a component is meeting its timing constraints. This might be realized in the form of an exception being raised, to indicate the failure of a component if its timing constraints are not met.
- *Timestamp*. As described in Chapter 4, the timestamp tactic is used to detect incorrect sequences of events, primarily in distributed message-passing systems. A timestamp can be established by assigning the state of a local clock to an event. Sequence numbers can also be used, since timestamps in a distributed system may be inconsistent across processors.
- *Condition Monitoring.* This tactic involves checking conditions in a process or device, or validating assumptions made during the design, perhaps by using assertions. Condition monitoring identifies system states that may lead to hazardous behavior.



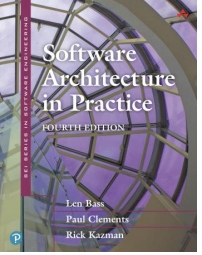
Unsafe State Detection

- *Sanity Checking.* The sanity checking tactic checks the validity or reasonableness of specific operation results, or inputs or outputs of a component. This tactic is typically based on a knowledge of the internal design, the state of the system, or the nature of the information under scrutiny.
- *Comparison*. The comparison tactic allows the system to detect unsafe states by comparing the outputs produced by a number of synchronized or replicated elements. Thus the comparison tactic works with a redundancy tactic. When the number of replicants is three or greater, the comparison tactic can not only detect an unsafe state but also indicate which component led to it.



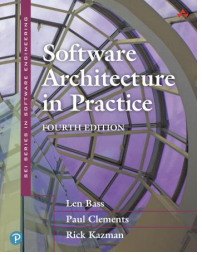
Containment - Redundancy

- *Replication.* Replication is the simplest redundancy tactic, as it just involves having clones of a com- ponent. Having multiple copies of identical components can be effective in protecting against random failures of hardware, but it cannot protect against design or implementation errors in hardware or software since there is no form of diversity embedded in this tactic.
- *Functional Redundancy*. Functional redundancy is intended to address the issue of common-mode failures (where replicas exhibit the same fault at the same time because they share the same implementation), by having design diversity. The outputs of functionally redundant components should be the same given the same input.
- *Analytic Redundancy*. Analytic redundancy involves partitioning the system into high assurance and high performance (low assurance) portions. The high assurance portion is designed to be simple and reliable, whereas the high performance portion is designed to be more complex and accurate, but less stable. If the high performance portion enters an unsafe state it can be replaced by the high assurance portion.



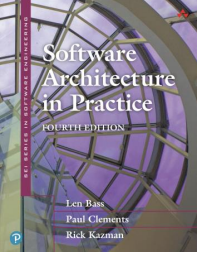
Containment – Limit Consequences

- *Abort.* If an operation is determined to be unsafe, it is aborted before it can cause damage.
- *Degradation*. The degradation tactic maintains the most critical system functions in the presence of component failures, dropping or replacing functionality in a controlled way. This approach allows individual component failures to gracefully reduce system functionality in a planned, deliberate, and safe way, rather than causing a complete system failure.
- *Masking*. The masking tactic masks a fault by comparing the results of several redundant components and employing a voting procedure in case one or more of the components differ.



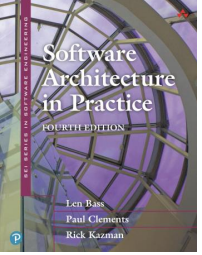
Containment – Barrier

- *Firewall.* The firewall tactic is a specific realization of the limit access tactic, which is described in Chapter 11. A firewall limits access to specified resources, typically processors, memory, and network connections.
- *Interlock*. The interlock tactic protects against failures arising from incorrect sequencing of events. Realizations of this tactic provide elaborate protection schemes by controlling all access to protected components, including controlling the correct sequencing of events affecting those components.



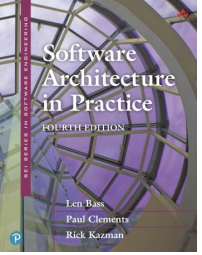
Recovery

- *Rollback.* The rollback tactic permits the system to revert to a saved copy of a previous known good state upon the detection of a failure. This tactic is often combined with checkpointing and transactions, to ensure that the rollback is complete and consistent.
- *Repair State*. The repair state tactic repairs an erroneous state—effectively increasing the set of states that a component can handle competently (i.e., without failure)—and then continues execution.
- *Reconfiguration*. Reconfiguration attempts to recover from failures by remapping the logical architecture onto the resources left functioning. Ideally, this remapping allows full functionality to be maintained. When this is not possible, the system may be able to maintain partial functionality in combination with the degradation tactic.



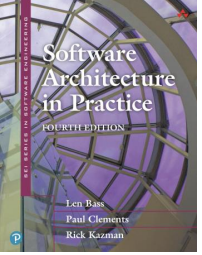
Tactics-Based Questionnaire for Safety

Tactics Group	Tactics Question	Supported? (Y/N)	Risk	Design Decisions and Location	Rationale and Assumptions
Unsafe State Avoidance	Do you employ **substitution**—that is, safer, often hardware-based protection mechanisms for potentially dangerous software design features?				
	Do you use a **predictive model** to predict the state of health of system processes, resources, or other properties—based on monitored information—not only to ensure that the system is operating within its nominal operating parameters, but also to provide early warning of a potential problem?				



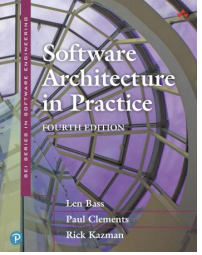
Tactics-Based Questionnaire for Safety

Unsafe State Detection	Do you use **timeouts** to determine whether the operation of a component meets its timing constraints?
	Do you use **timestamps** to detect incorrect sequences of events?
	Do you employ **condition monitoring** to check conditions in a process or device, particularly to validate assumptions made during design?
	Is **sanity checking** employed to check the validity or reasonableness of specific operation results, or inputs or outputs of a component?
	Does the system employ **comparison** to detect unsafe states, by comparing the outputs produced based on the number of synchronized or replicated elements?



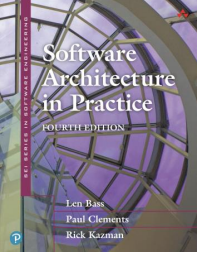
Tactics-Based Questionnaire for Safety

Containment: Redundancy	Do you use **replication**—clones of a component—to protect against random failures of hardware?
	Do you use **functional redundancy** to address the common-mode failures by implementing diversely designed components?
	Do you use **analytic redundancy**—functional "replicas" that include high assurance/high performance and low assurance/low performance alternatives—to be able to tolerate specification errors?



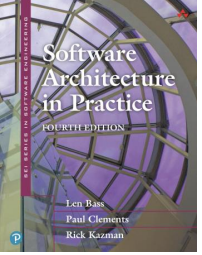
Tactics-Based Questionnaire for Safety

Containment: Limit Consequences	Can the system **abort** an operation that is determined to be unsafe before it can cause damage?
	Does the system provide controlled **degradation**, where the most critical system functions are maintained in the presence of component failures, while less critical functions are dropped or degraded?
	Does the system **mask** a fault by comparing the results of several redundant components and employ a **voting** procedure in case one or more of the components differ?



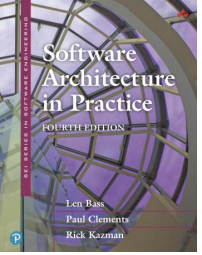
Tactics-Based Questionnaire for Safety

Containment: Barrier	Does the system support limiting access to critical resources (e.g., processors, memory, and network connections) through a **firewall**?
	Does the system control access to protected components and protect against failures arising from incorrect sequencing of events through **interlocks**?
Recovery	Is the system able to **roll back**—that is, to revert to a previous known good state—upon the detection of a failure?
	Can the system **repair a state** determined to be erroneous, without failure, and then continue execution?
	Can the system **reconfigure** resources, in the event of failures, by remapping the logical architecture onto the resources left functioning?



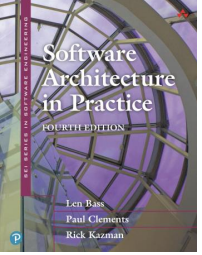
Redundant Sensors Pattern for Safety

- If the data produced by a sensor is important to determine whether a state is safe or unsafe, that sensor should be replicated. This protects against the failure of any single sensor.
- Also, independent software should monitor each sensor.



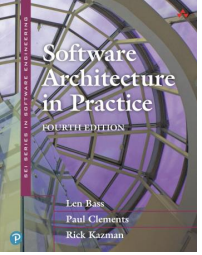
Redundant Sensors Pattern Benefits

- Benefits:
  - This form of redundancy, which is applied to sensors, guards against the failure of a single sensor.



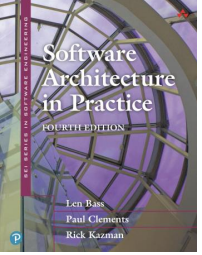
Redundant Sensors Pattern Tradeoffs

- Tradeoffs:
  - Redundant sensors add cost to the system, and processing the inputs from multiple sensors is more complicated than processing the input from a single sensor.



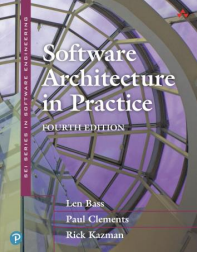
Monitor-Actuator Pattern for Safety

- This pattern focuses on two software elements—a monitor and an actuator controller—that are employed before sending a command to a physical actuator.
- The actuator controller performs the calculations necessary to determine the values to send to the physical actuator.
- The monitor checks these values for reasonableness before sending them.
- This separates the computation of the value from the testing of the value.



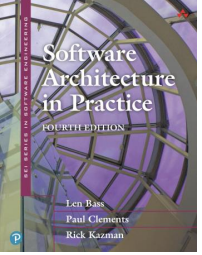
Monitor-Actuator Pattern Benefits

- In this form of redundancy applied to actuator control, the monitor acts as a redundant check on the actuator controller computations.



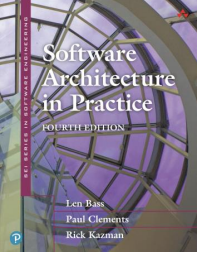
Monitor-Actuator Pattern Tradeoffs

- The development and maintenance of the monitor take time and resources.
- Because of the separation between actuator control and monitoring, this tradeoff is easy to manage by making the monitor as simple (easy to produce but may miss errors) or as sophisticated (more complex but catches more errors) as required.



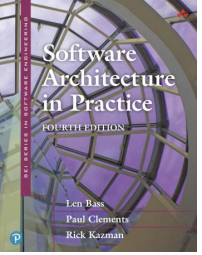
Separated Safety Pattern for Safety

- Safety-critical systems must frequently be certified as safe by some authority.
- Certifying a large system is expensive, but dividing a system into safety-critical portions and non-safety-critical portions can reduce those costs. Only the safety-critical portion must be certified.
- Likewise, the division into safety-critical and non-critical portions must be certified to ensure that there is no influence on the safety-critical portion from the non-safety-critical portion.



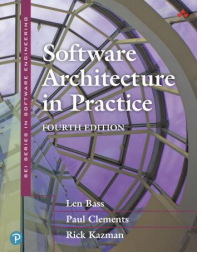
Separated Safety Pattern Benefits

- The cost of certifying the system is reduced because you need to certify only a (usually small) portion of the total system.
- Cost and safety benefits accrue because the effort focuses on just those portions of the system that are germane to safety.



Separated Safety Pattern Tradeoffs

- The work involved in performing the separation can be expensive. However, this approach limits the risk and consequences of bugs in the non-safety-critical portion from affecting the safety-critical portion.
- Separating the system and convincing the certification agency that the separation was performed correctly and that there are no influences from the non-safety-critical portion on the safety-critical portion is difficult, but is easier than the alternative: having the agency certify everything to the same rigid level.



Summary

- Safety is about a system's ability to avoid straying into states that lead to damage, injury, or loss of life.
- Design for safety begins with identifying safety-critical functions using techniques like FMEA and FTA.
- Designing for safety is challenging but there is a rich set of tactics and patterns to aid the designer.