

1. Funciones del Arquitecto, FOSA
2. Atributos de Calidad, FOSA y SEI.
3. Tácticas para Disponibilidad (Availability), Rendimiento (Performance) y Despliegue (Deployability). SEI.
4. SLA, SLI, SLO o KPI. SEI y otros.
5. Estilo vs Diseño, FOSA
6. Clasificación de Estilos: por Despliegue: Monolitos vs Sistemas Distribuidos. SAP Report.
7. Listado de Estilos. FoSA, Azure
8. Listado de Patrones seleccionados. SAP Report, Azure

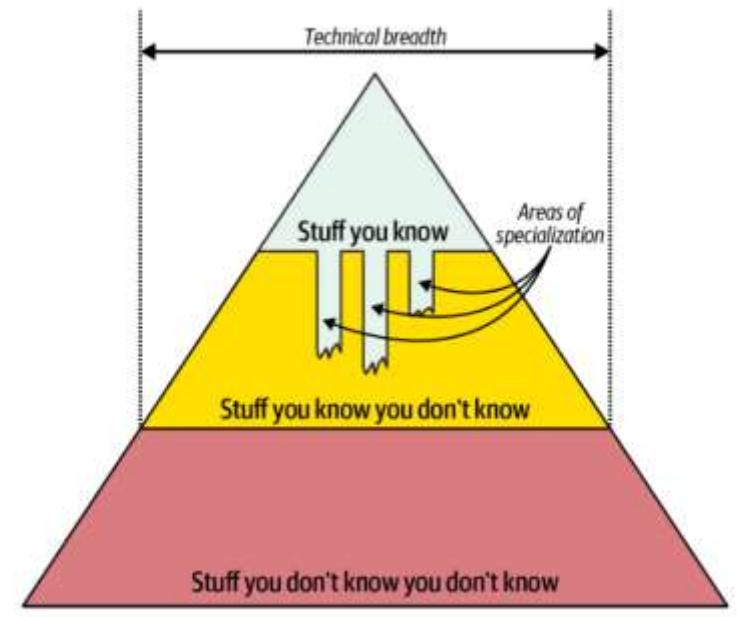
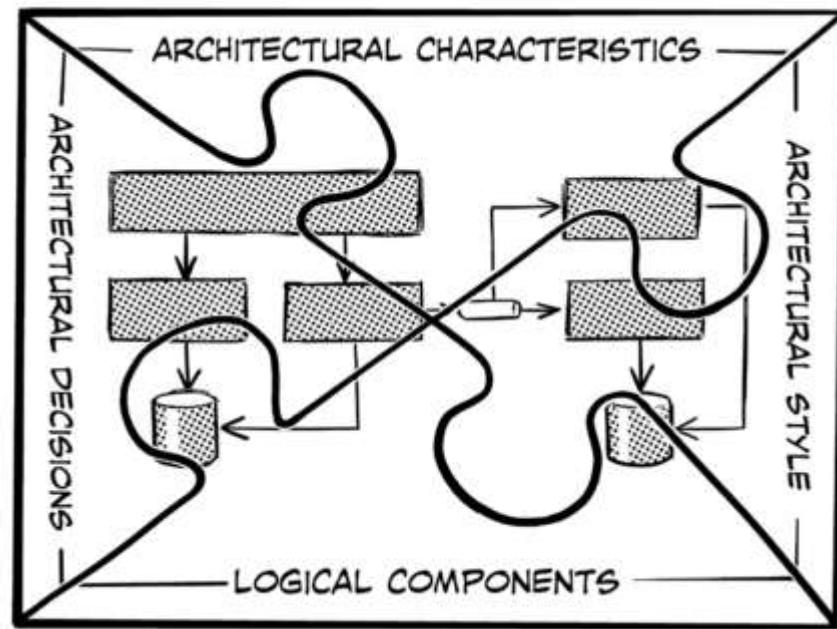


Figure 2-5. Enhanced breadth and shrinking depth for the architect role

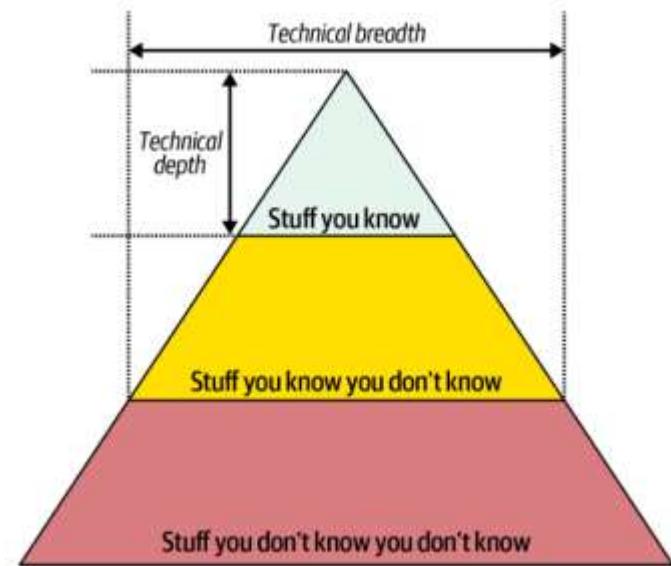


Figure 2-4. How much someone knows about a topic in technical depth, and how many topics someone knows is technical breadth

*Everything in software architecture is a trade-off.*

—First Law of Software Architecture

*Why is more important than how.*

—Second Law of Software Architecture

*Most architecture decisions aren't binary but rather exist on a spectrum between extremes.*

—Third Law of Software Architecture

- Make architecture decisions
- Continually analyze the architecture
- Keep current with latest trends
- Ensure compliance with decisions
- Understand diverse technologies, frameworks, platforms, and environments
- Know the business domain
- Lead a team and possess interpersonal skills
- Understand and navigate organizational politics

## Dice modularidad en Prin

- > Preface
- ✓ 1. Introduction
  - Defining Software Architecture
  - Laws of Software Architecture
  - ✓ Expectations of an Architect
    - Make Architecture Decisions
    - Continually Analyze the Architecture
    - Keep Current with Latest Trends
    - Ensure Compliance with Decisions
    - Understand Diverse Technologies
    - Know the Business Domain
    - Possess Interpersonal Skills
    - Understand and Navigate Politics
- Roadmap
- I. Foundations
- ✓ 2. Architectural Thinking
  - ✓ Architecture Versus Design
    - Strategic Versus Tactical Decisions
    - Level of Effort
    - The Significance of Trade-Offs
  - ✓ Technical Breadth
    - The 20-Minute Rule
    - Developing a Personal Radar
    - Analyzing Trade-Offs
    - Understanding Business Drivers
    - Balancing Architecture and Hands-On Coding
  - There's More to Architectural Thinking
- ✓ 3. Modularity
  - Modularity Versus Granularity
  - Defining Modularity
  - > Measuring Modularity
  - From Modules to Components
- ✓ 4. Architectural Characteristics Defined
  - Architectural Characteristics and System Design
  - > Architectural Characteristics (Partially) Listed
  - Trade-Offs and Least Worst Architecture
- > Summary of Architectural Characteristics

## Chapter 3. Modularity

Architects and developers have struggled with the concept of modularity for quite some time, as is evident in this quote from *Computer Architecture: Art and Science* (Van Nostrand Reinhold, 1978):

95% of the words [written about software architecture] are extolling the benefits of “modularity” and little, if anything, about how to achieve it.

—Glenford J.

# Arch Characteristics/Quality attributes

ISO 25010:2023, SAIP

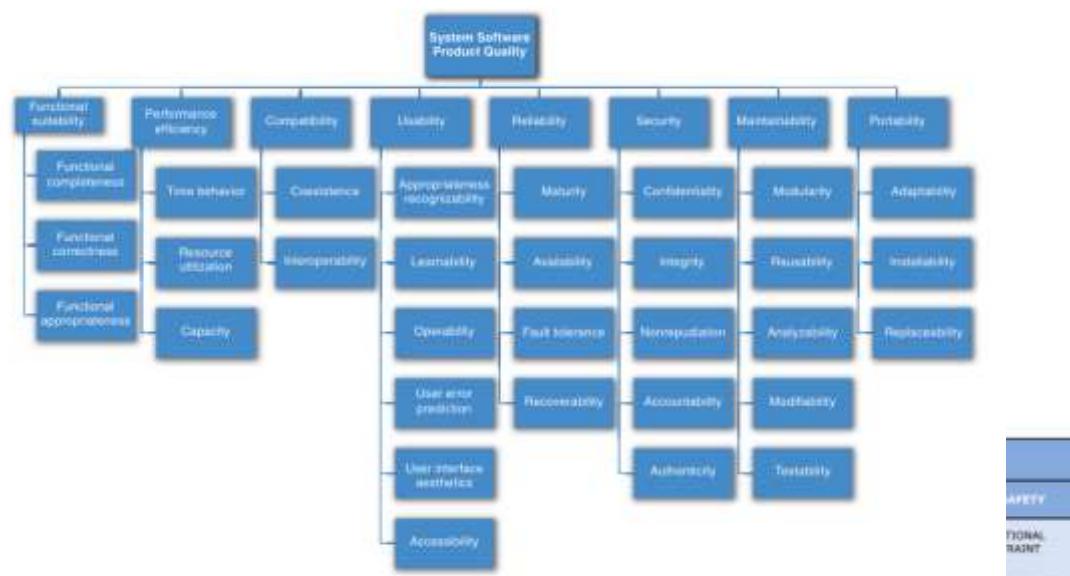
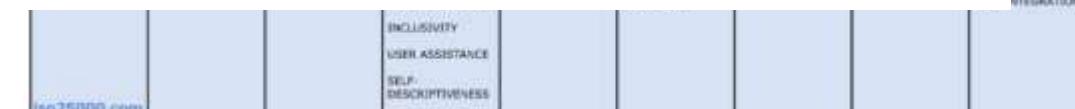


FIGURE 14.1 ISO/IEC FCD 25010 Product Quality Standard



Even though to enterprise interoperability, but regulations/industries standard/market velocity/etc.

Operational ACh	Structural ACh	Cloud Ch	Cross-Cutting ACh
Term	Term	Term	Term
<b>FoSA. Ch 4</b> <b>Architectur al Characterist ics Defined</b>	Availability	Configurability	On-demand scalability
Continuity	Extensibility	On-demand elasticity	Archivability
Performance	Installability	Zone-based availability	Authentication
Recoverability	Leverageability/reuse	Region-based privacy and security	Authorization
Reliability/safety	Localization	Maintainability	Legal
Robustness	Portability	Upgradeability	Privacy
Scalability			Security
			Supportability
			Usability/achievability

HLD

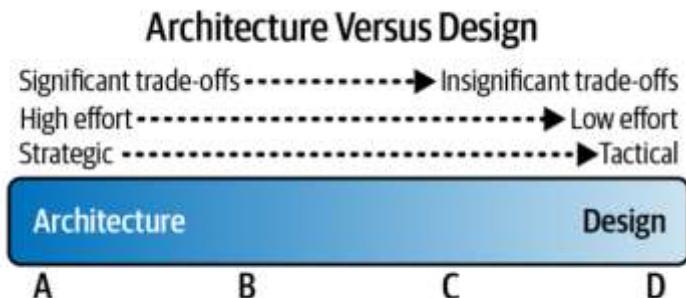


Figure 2-1. The spectrum between architecture and design

Who needs an architect – Fowler, who is architect vs Design: [Link](#)

## **Effort**

*Architecture is the stuff you can't Google or ask an LLM about.*

—Mark Richards

*There are no right or wrong answers in architecture—only trade-offs.*

—Neal Ford

choice. However, to quote [Rich Hickey](#), the creator of the Clojure programming language:

*Programmers know the benefits of everything and the trade-offs of nothing. Architects need to understand both.*

—Rich Hickey

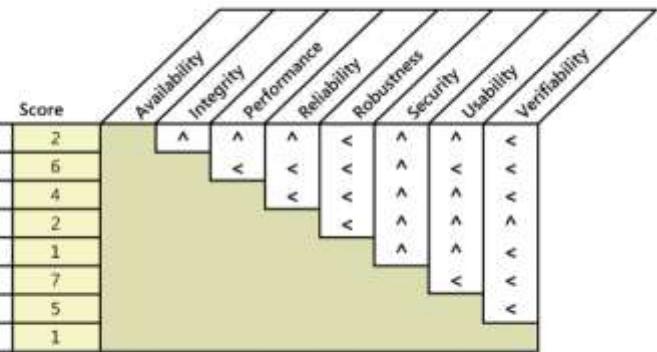


Figure 14-1. Sample quality attribute prioritization for an airport check-in kiosk.

	Avail	Effic	Insta	Integ	Inter	Modifiability	Performance	Portability	Reliability	Reusability	Robustness	Safety	Scalability	Security	Usability	Verifiability
Availability									+	+						
Efficiency	+			-	-	+	-			-	+	-				
Installability	+							+				+				
Integrity		-		-			-		-	+		+	-	-		
Interoperability	+	-	-	-			-	+	+	+	-		-			
Modifiability	+	-					-	+	+		-		-			
Performance	+			-	-		-									
Portability		-		+	-	-				+						
Reliability	+	-		+		+	-					+				
Reusability		-		-	+	+	-	+								
Robustness	+	-	+	+	+		-		+							
Safety		-		+	+		-				+					
Scalability	+	+	+				+	+	+		+					
Security	+		+	+			-	-	+		+					
Usability		-	+				-	-	+		+					
Verifiability	+		+	+		+			+	+	+					

Figure 14-2. Positive and negative relationships among selected quality attributes.

## SWR, Ch14, - Quality attribute trade-offs

Step 1: Start with a broad taxonomy

Step 2: Reduce the list

Step 3: Prioritize the attributes

Step 4: Elicit specific expectations for each

Step 5: Specify well-structured quality requirements

Table 14-5. Translating quality attributes into technical specifications

Quality attributes	Likely technical information category
Installability, integrity, interoperability, reliability, robustness, safety, security, usability, verifiability	Functional requirement
Availability, efficiency, modifiability, performance, reliability, scalability	System architecture
Interoperability, security, usability	Design constraint
Efficiency, modifiability, portability, reliability, reusability, scalability, verifiability	Design guideline

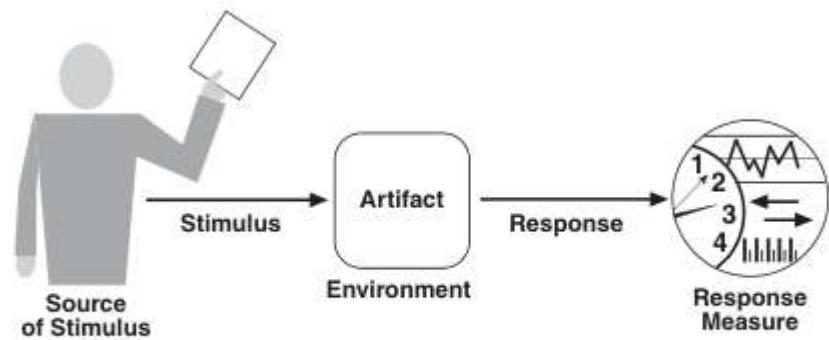


FIGURE 4.1 The parts of a quality attribute scenario

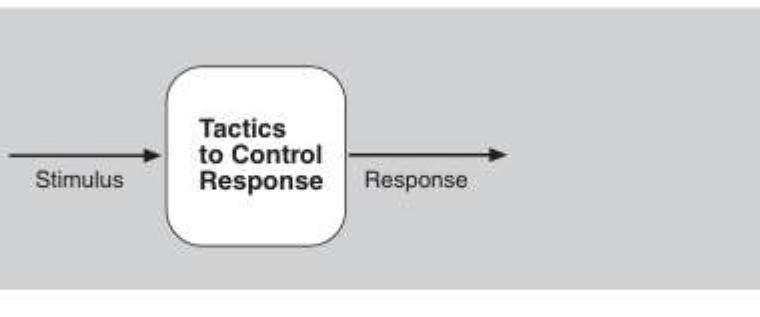


FIGURE 4.3 Tactics are intended to control responses to stimuli.

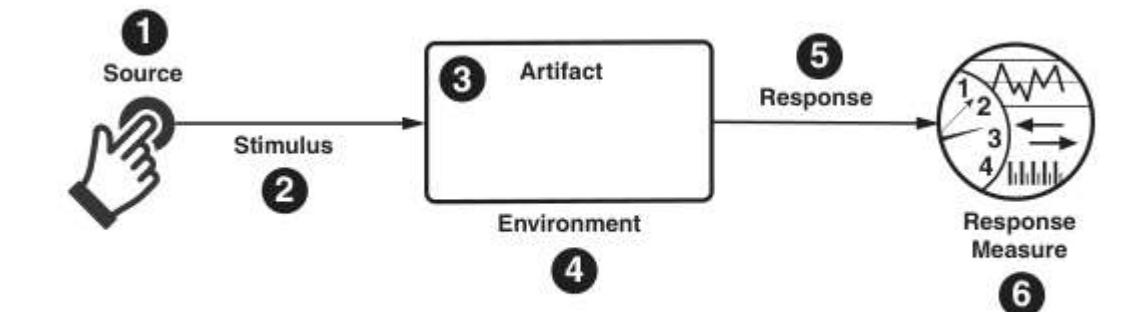


FIGURE 3.1 The parts of a quality attribute scenario

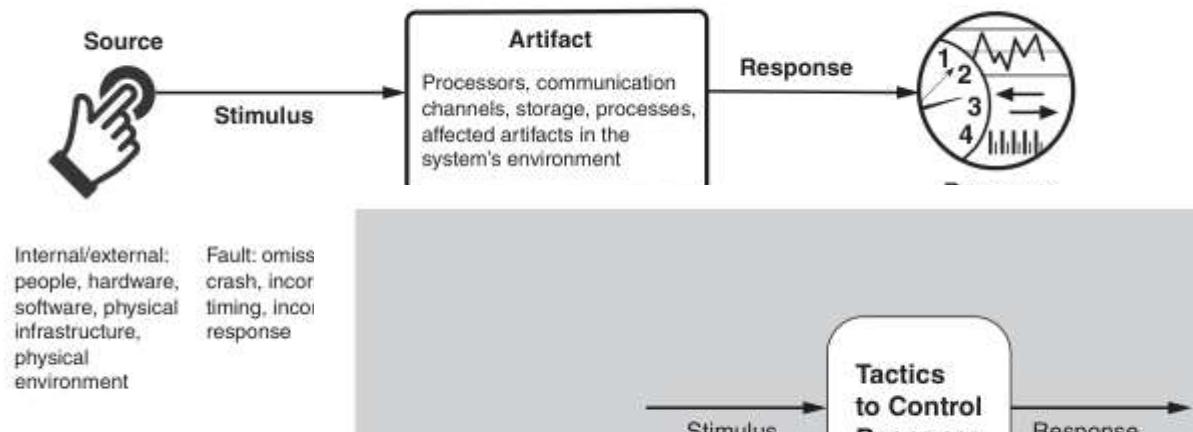


FIGURE 3.2 A general scenario

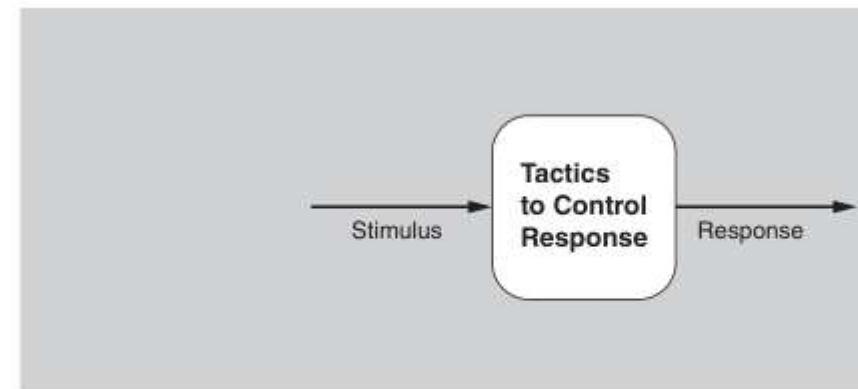


FIGURE 3.3 Tactics are intended to control responses to stimuli.

## 3rd Edicion

- PART TWO: QUALITY ATTRIBUTES
  - > · 4 Understanding Quality Attributes
  - > · 5 Availability
  - > · 6 Interoperability
  - > · 7 Modifiability
  - > · 8 Performance
  - > · 9 Security
  - > · 10 Testability
  - > · 11 Usability
  - > · 12 Other Quality attributes
  - > · 13 Architectural Tactics and Patterns
  - > · 14 Quality Attribute Modeling and Analysis

## 4th Edition

- PART II: QUALITY ATTRIBUTES
  - > · CHAPTER 3 Understanding Quality Attributes
  - > · CHAPTER 4 Availability
  - > · CHAPTER 5 Deployability
  - > · CHAPTER 6 Energy Efficiency
  - > · CHAPTER 7 Integrability
  - > · CHAPTER 8 Modifiability
  - > · CHAPTER 9 Performance
  - > · CHAPTER 10 Safety
  - > · CHAPTER 11 Security
  - > · CHAPTER 12 Testability
  - > · CHAPTER 13 Usability
  - > · CHAPTER 14 Working with Other Quality Attributes

- Availability
- Modifiability
- Performance
- Security
- Testability
- Usability

# 3<sup>rd</sup> Edition

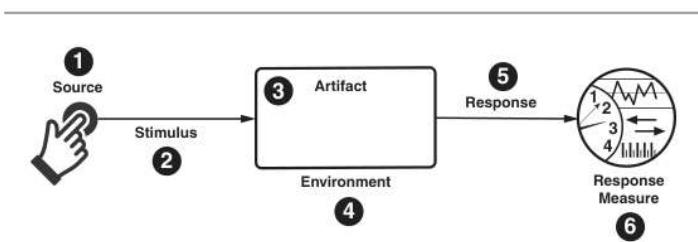


FIGURE 3.1 The parts of a quality attribute scenario

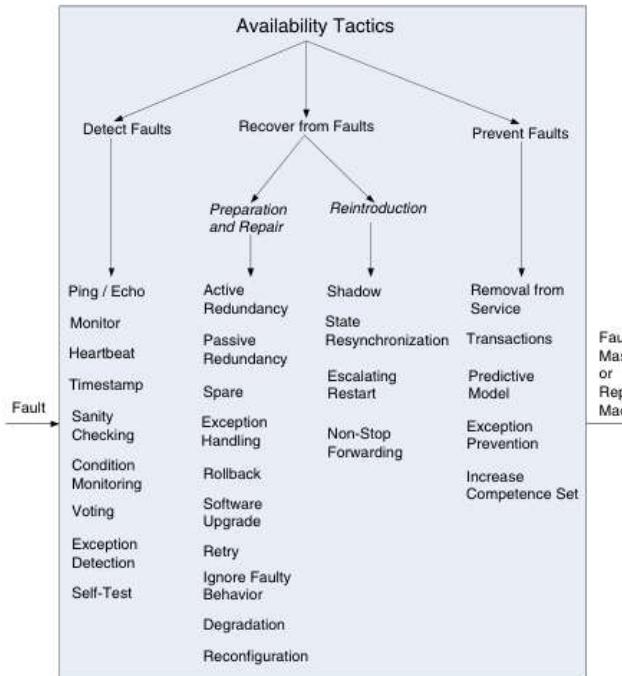


FIGURE 5.5 Availability tactics

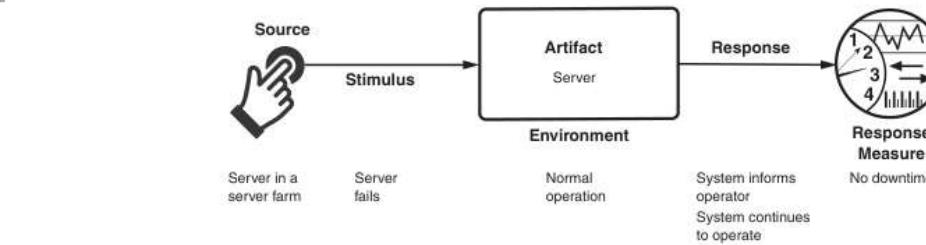


FIGURE 4.1 Sample concrete availability scenario

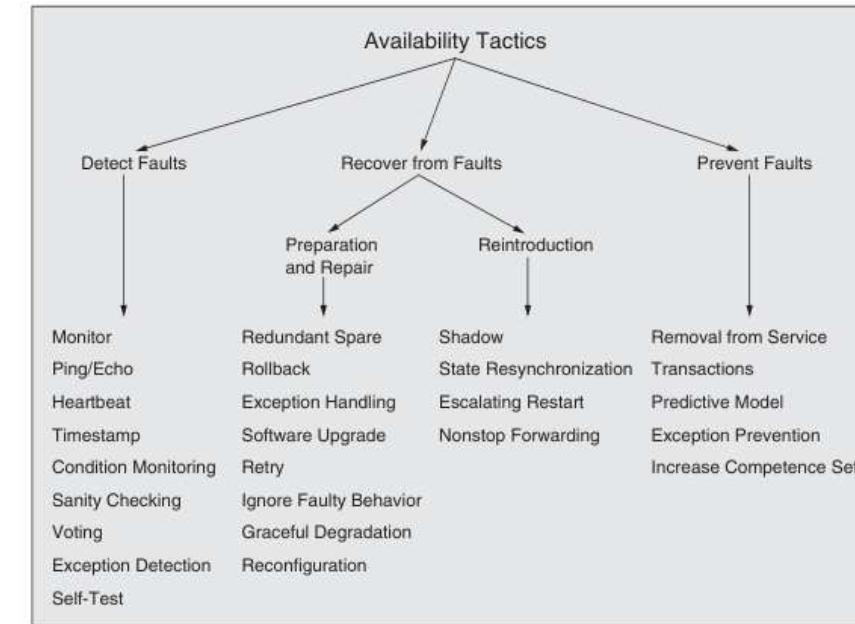


FIGURE 4.3 Availability tactics

# 4<sup>th</sup> Edition

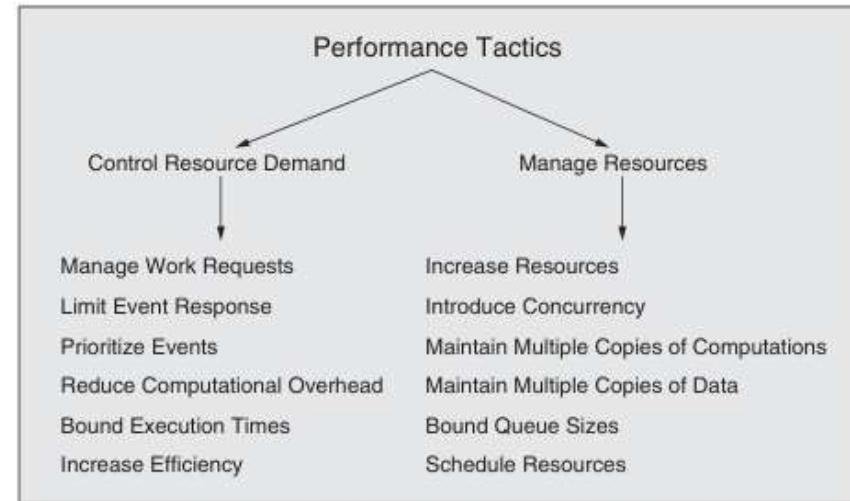


FIGURE 9.3 Performance tactics