

Cyber-physical systems

ME 2150 HACPS

What Are Cyber-Physical Systems?

- ▶ **Definition:** Integrations of computational and physical components designed to interact with and control the physical world.
- ▶ **Core Components:**
 - ▶ Embedded software
 - ▶ Sensors and actuators
 - ▶ Communication networks
- ▶ **Examples:**
 - ▶ Smart thermostats
 - ▶ Autonomous vehicles
 - ▶ Power grids
 - ▶ Medical devices
 - ▶ Industrial automation

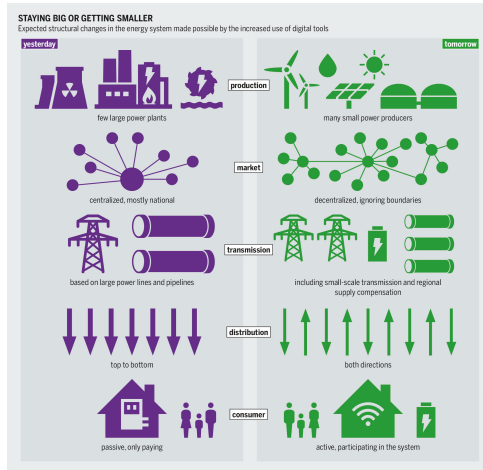


Figure 1: Smart Grid (*source: Wikipedia*)

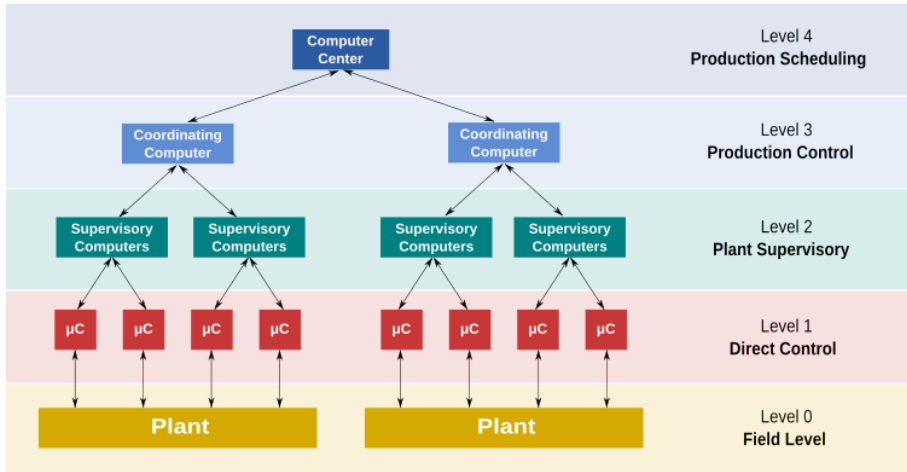


Figure 2: Industrial control system (*source*: Wikipedia)



Figure 3: Infusion pump (*source: Wikipedia*)



Figure 4: Programmable logic controllers (*source: Wikipedia*)

Key Characteristics of CPS

- ▶ Tight integration of:
 - ▶ Computation
 - ▶ Communication
 - ▶ Control
- ▶ Features:
 - ▶ Real-time data collection and processing
 - ▶ Feedback loops for adaptation and optimization
 - ▶ Precision and responsiveness
- ▶ Applications:
 - ▶ Stabilizing aircraft
 - ▶ Robotic manufacturing
 - ▶ Patient monitoring in healthcare

Benefits and Challenges

- ▶ Benefits:
 - ▶ Enhanced efficiency and automation
 - ▶ Safer infrastructure
 - ▶ Optimized energy use
 - ▶ Innovations in healthcare
- ▶ Challenges:
 - ▶ Dependability and fault tolerance
 - ▶ Cybersecurity risks
 - ▶ Ethical and regulatory concerns

Examples of CPS Failures

Event/Failure	Year
The Therac-25 Incident	1980s
Patriot Missile Failure	1991
The Ariane 5 Explosion	1996
Mars Climate Orbiter Loss	1999
Northeast Blackout	2003
Toyota Unintended Acceleration	2009—2010
Stuxnet Malware Attack	2010
Boeing 737 MAX Crashes	2018

Ariane 5: Dead code with no protection against integer overflow caused improper exception handling, which halted the INS.

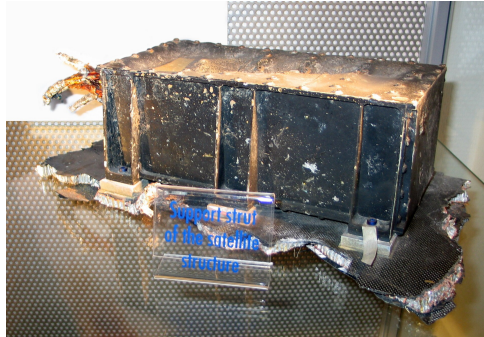


Figure 5: Ariane 5 fragment (*source: Wikipedia*)

Mars Climate Orbiter: There was mismatch between units: SI units by NASA and US customary units by Lockheed Martin

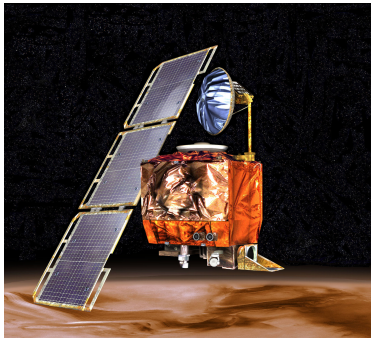


Figure 6: Mars climate orbiter (*source:*) Wikipedia

Failures in a CPS can occur at various levels and involve both computational and physical components.

1. Software Failures

- ▶ Bugs and Logic Errors: Flaws in code causing incorrect system behavior.
- ▶ Concurrency Issues: Errors from improper handling of parallel processes (e.g., race conditions).
- ▶ Inadequate Testing: Lack of stress testing or real-world scenario validation.
- ▶ Legacy Code Issues: Reuse of outdated or unsuitable software without adaptation.
- ▶ Algorithmic Errors: Inaccurate computations or decisions based on incorrect algorithms.

2. Hardware Failures

- ▶ Sensor Malfunctions: Faulty or degraded sensors providing incorrect data.
- ▶ Actuator Failures: Actuators unable to perform physical actions correctly.
- ▶ Communication Failures: Loss or corruption of signals between components.
- ▶ Power Supply Issues: Interruptions or inconsistencies in power sources.
- ▶ Wear and Tear: Physical degradation over time in mechanical or electronic components.

Failures in CPS *cont'*

3. Integration Failures

- ▶ System-Level Incompatibility: Mismatch between hardware, software, or network protocols.
- ▶ Interdisciplinary Misalignment: Miscommunication between teams in software, hardware, and system integration.
- ▶ Model Inaccuracy: Computational models not accurately reflecting physical processes.
- ▶ Interference: Crosstalk or unintended interactions between interconnected systems.

4. Real-Time Performance Failures

- ▶ Timing Issues: Delayed responses failing to meet real-time requirements.
- ▶ Resource Starvation: Insufficient computational or memory resources during critical tasks.
- ▶ Clock Synchronization Failures: Lack of alignment in timing between components.

Failures in CPS *cont'*

5. Communication and Network Failures

- ▶ Latency: Delays in data transmission impacting real-time decision-making.
- ▶ Data Loss: Missing information due to packet drops or communication errors.
- ▶ Signal Jamming: Intentional or unintentional disruption of wireless signals.
- ▶ Bandwidth Limitations: Insufficient capacity for data-intensive operations.

6. Cybersecurity Failures

- ▶ Hacking and Intrusion: Unauthorized access to system components.
- ▶ Malware and Viruses: Malicious software disrupting operations (e.g., Stuxnet).
- ▶ Data Breaches: Compromise of sensitive or critical data.
- ▶ Denial of Service (DoS) Attacks: Overloading the system with excessive requests.

Failures in CPS *cont'*

7. Environmental and Physical Failures

- ▶ External Environmental Factors:
 - ▶ Extreme temperatures
 - ▶ Vibration or shock
 - ▶ Electromagnetic interference (EMI)
 - ▶ Natural disasters (e.g., floods, earthquakes)
- ▶ Physical Damage:
 - ▶ Accidental impacts
 - ▶ Tampering or vandalism

8. Human Factors

- ▶ Design Errors: Poorly planned interfaces, workflows, or system architecture.
- ▶ Operational Errors: Mistakes made by users or operators during interaction with the system.
- ▶ Mismanagement: Insufficient oversight, training, or maintenance.

Failures in CPS *cont'*

9. Systemic Failures

- ▶ Cascading Failures: A single fault triggering failures in interconnected systems.
- ▶ Feedback Loop Errors: Faulty feedback causing instability or incorrect adjustments.
- ▶ Control Loop Failures: Inability of control systems to stabilize or maintain desired states.

10. Regulatory and Ethical Failures

- ▶ Non-Compliance: Failure to adhere to safety, quality, or industry standards.
- ▶ Ethical Concerns: Neglect of user safety, privacy, or fairness in system design.

Lessons from CPS Failures

1. Software Failures: Use formal verification and rigorous testing to prevent bugs and logic errors.
2. Hardware Failures: Implement redundancy and regular maintenance for reliability.
3. Integration Failures: Conduct thorough system-level testing and ensure interdisciplinary collaboration.
4. Human Factors: Design user-friendly systems and provide comprehensive operator training.
5. Communication Failures: Build robust, fail-safe communication protocols.

Lessons from CPS Failures *cont'd*

6. Cybersecurity Failures: Integrate strong security measures and regularly update systems.
7. Real-Time Performance Failures: Ensure adequate resources and precise timing for real-time operations.
8. Environmental Failures: Design systems to withstand environmental extremes and physical shocks.
9. Systemic Failures: Prevent cascading effects with robust, fault-tolerant designs.
10. Regulatory Failures: Adhere to industry standards and incorporate ethical considerations.

High-Assurance systems are designed with rigorous reliability, safety, and security for critical tasks in scenarios with high consequences.

1. Safety: The system must operate without causing harm to people or the environment under all conditions.
2. Security: Strong measures must protect the system against unauthorized access, tampering, and cyber threats.
3. Reliability: The system must consistently perform its intended functions as designed.
4. Robustness: The system should gracefully handle unexpected inputs, errors, or adverse conditions.
5. Fail-Safe Design: In the event of a critical failure, the system must default to a safe and stable state.

High-Assurance systems are designed with rigorous reliability, safety, and security for critical tasks in scenarios with high consequences.

6. Real-Time Responsiveness: The system must meet strict timing requirements for tasks in real-world scenarios.
7. Fault Tolerance: The system must maintain functionality even when components fail or degrade.
8. Formal Verification: Mathematical models should validate that the system operates correctly and adheres to its specifications.
9. Comprehensive Testing: Rigorous testing must address edge cases, stress conditions, and real-world scenarios.
10. Standards Compliance: The system must adhere to applicable industry standards and certifications to ensure safety and quality.

Procedure for Ensuring High-Assurance

1. Define Requirements and Constraints
 - ▶ Document functional, safety, and security requirements.
 - ▶ Specify constraints and identify failure modes.
2. Risk Analysis and Hazard Assessment
 - ▶ Use methods like FMEA, HAZOP, or STPA.
 - ▶ Prioritize high-risk scenarios.
3. Formal Design and Architecture
 - ▶ Modular, fault-tolerant designs with redundancy.
 - ▶ Develop system models for verification.
4. Formal Verification
 - ▶ Use mathematical proofs for correctness.
5. Comprehensive Testing
 - ▶ Include unit, integration, and real-world scenario testing.

Procedure for Ensuring High-Assurance *cont'*

6. Implement Standards and Best Practices
 - ▶ Adhere to industry standards
7. Cross-Disciplinary Collaboration
 - ▶ Involve experts across relevant fields.
8. Documentation and Traceability
 - ▶ Maintain detailed records and ensure traceability.
9. Continuous Monitoring and Feedback
 - ▶ Implement real-time monitoring and update systems as needed.
10. Lifecycle Assurance
 - ▶ Reassess and safely decommission systems.