CS42600: Computer Security

Assurance

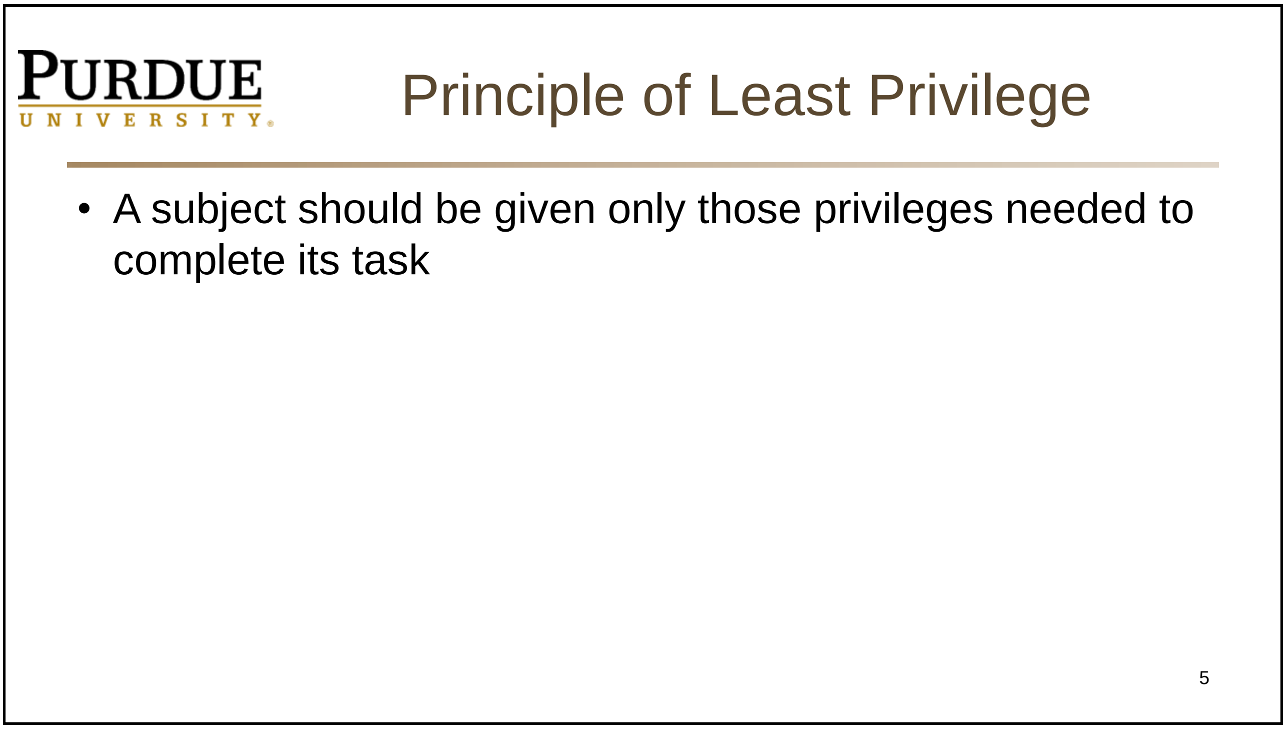
Prof. Chris Clifton

11 April 2019

Center for Education and Research in Information Assurance and Security

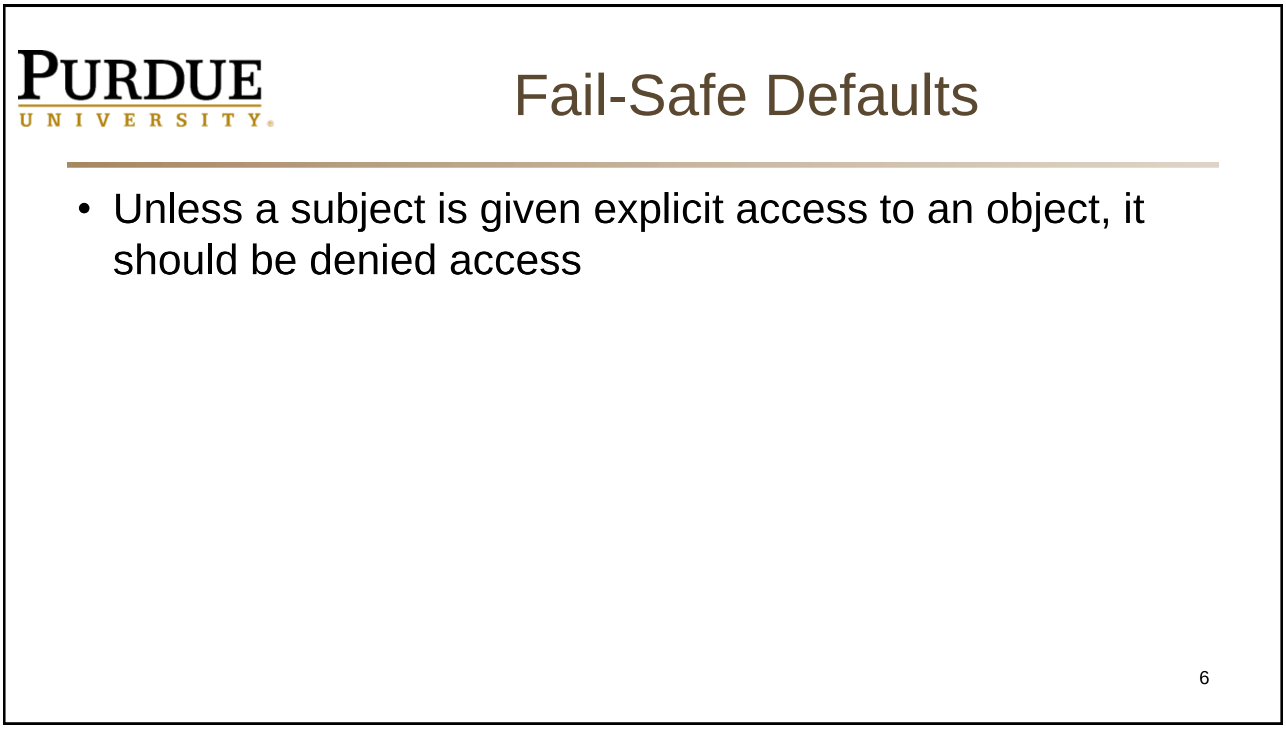
Building Real Systems

- Theory allows formal proof of known security policies
  - For components
  - And collections of components
- What should the security policies be?
- Design principles:
  - Guiding standards that are relatively independent of policy



Principle of Least Privilege

- A subject should be given only those privileges needed to complete its task



Fail-Safe Defaults

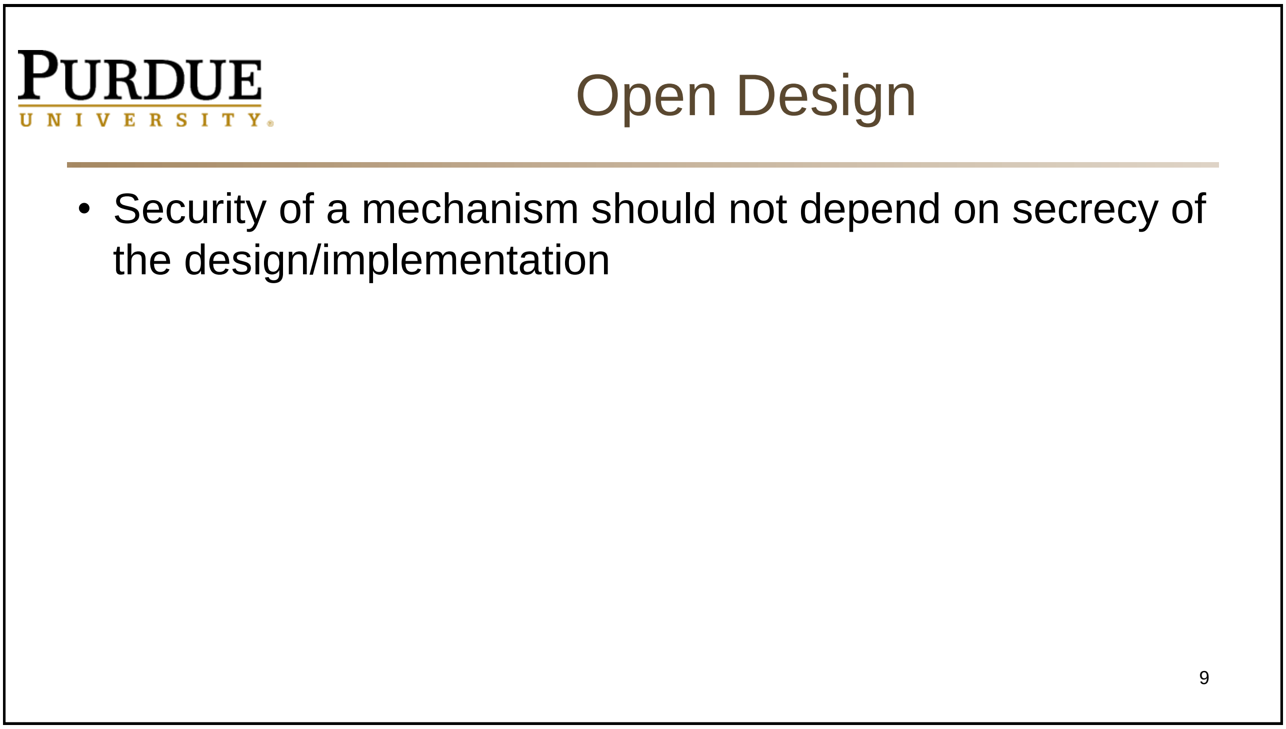
- Unless a subject is given explicit access to an object, it should be denied access

Economy of Mechanism

- Security mechanisms should be as simple as possible
  - *But no simpler*

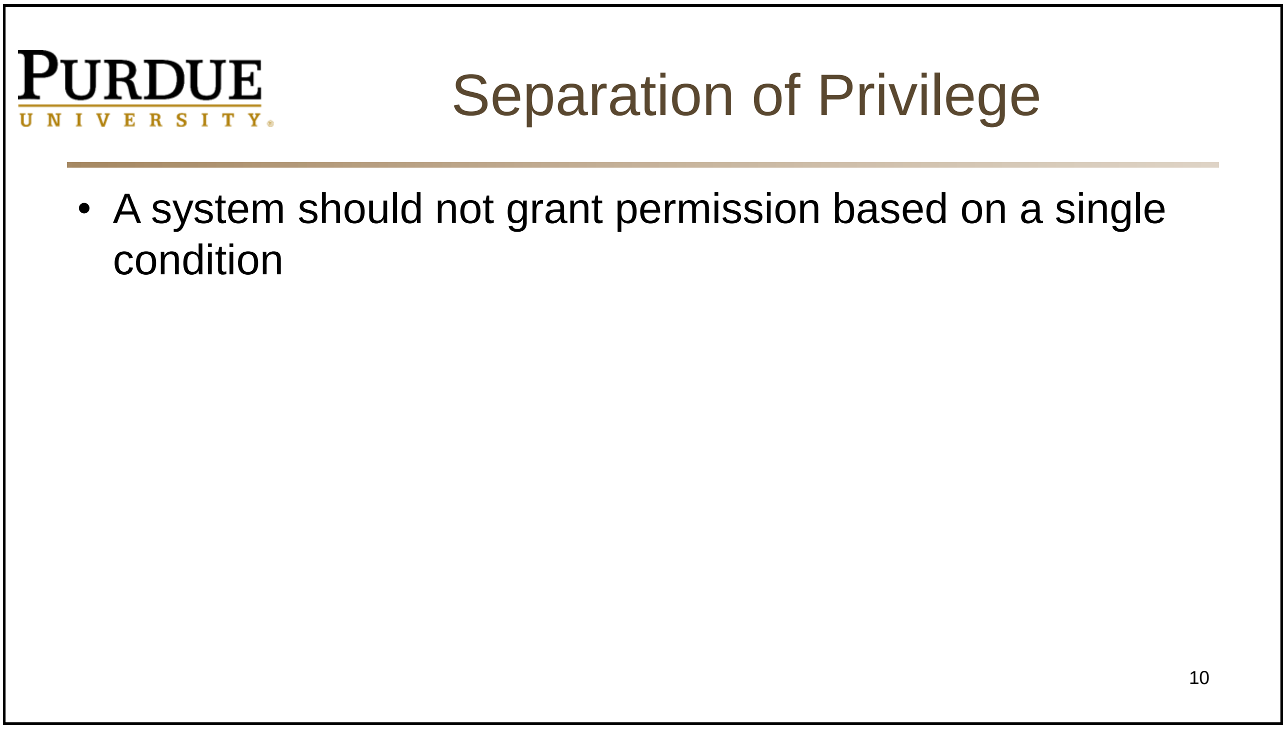
Complete Mediation

- All accesses to an object must be checked to ensure they are allowed



Open Design

- Security of a mechanism should not depend on secrecy of the design/implementation



Separation of Privilege

- A system should not grant permission based on a single condition

Least Common Mechanism

- Mechanisms used to access resources should not be shared

Psychological Acceptability

- Security mechanisms should not make the resource more difficult to access than if security mechanisms were not present

PURDUE
UNIVERSITY

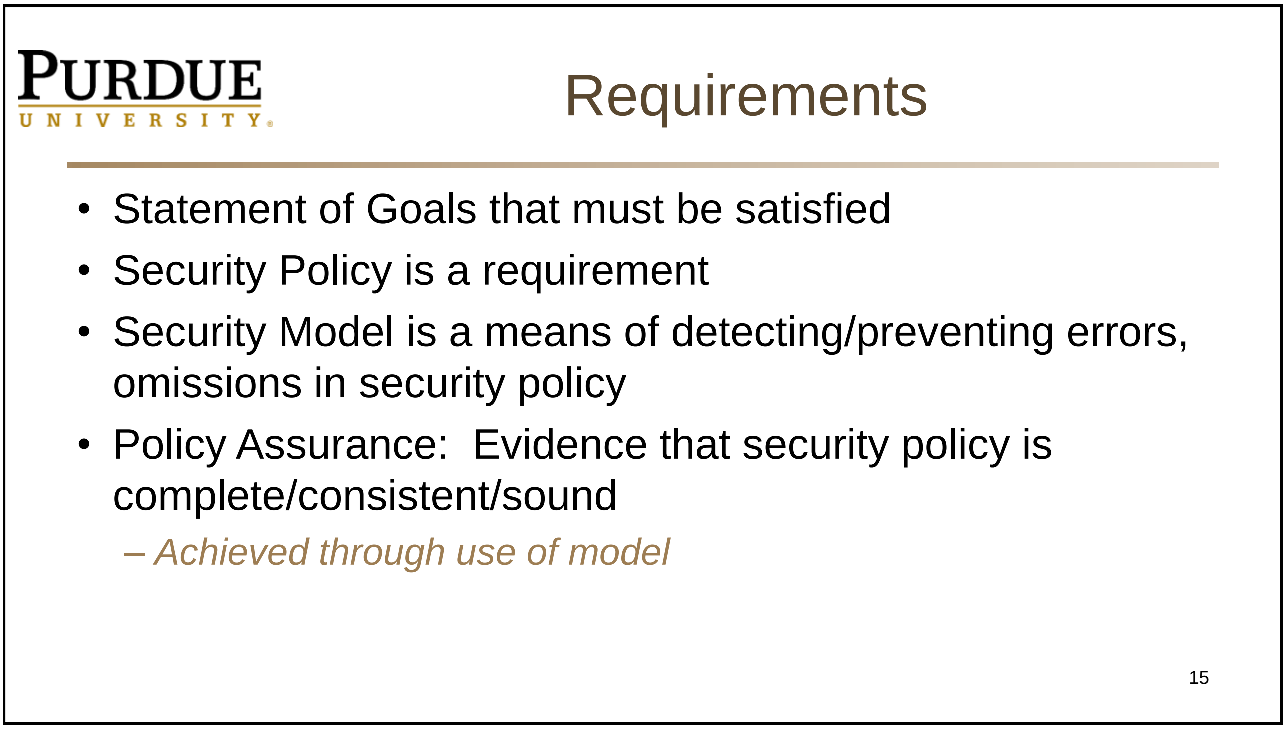
Secure Systems in Practice

- Formal verifications of entire systems *not yet* a practice
  - So what do we mean by secure?
- Trustworthy: Sufficient evidence to believe system will meet requirements
  - How do we measure this?
- Assurance: Confidence a system meets security requirements
  - Often based on development processes
- Trusted System: Evaluated / passed in terms of well-defined requirements, evaluation methods

PURDUE
UNIVERSITY

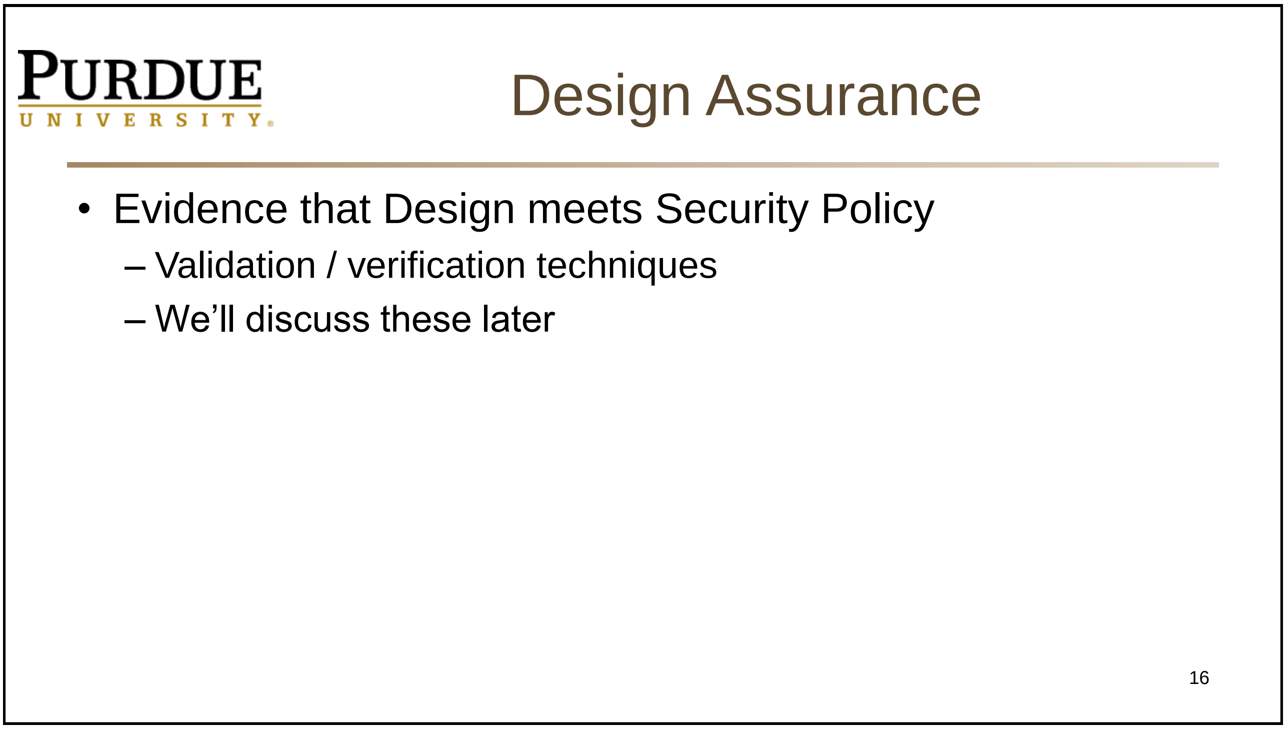
High-Assurance Development Methodologies Control:

- Requirements definitions, omissions, mistakes
- System design flaws
- Hardware implementation flaws
- Software implementation errors
- System use/operation errors
- Willful system misuse
- Hardware malfunction
- Natural / environmental effects
- Evolution/maintenance/upgrades/decommission



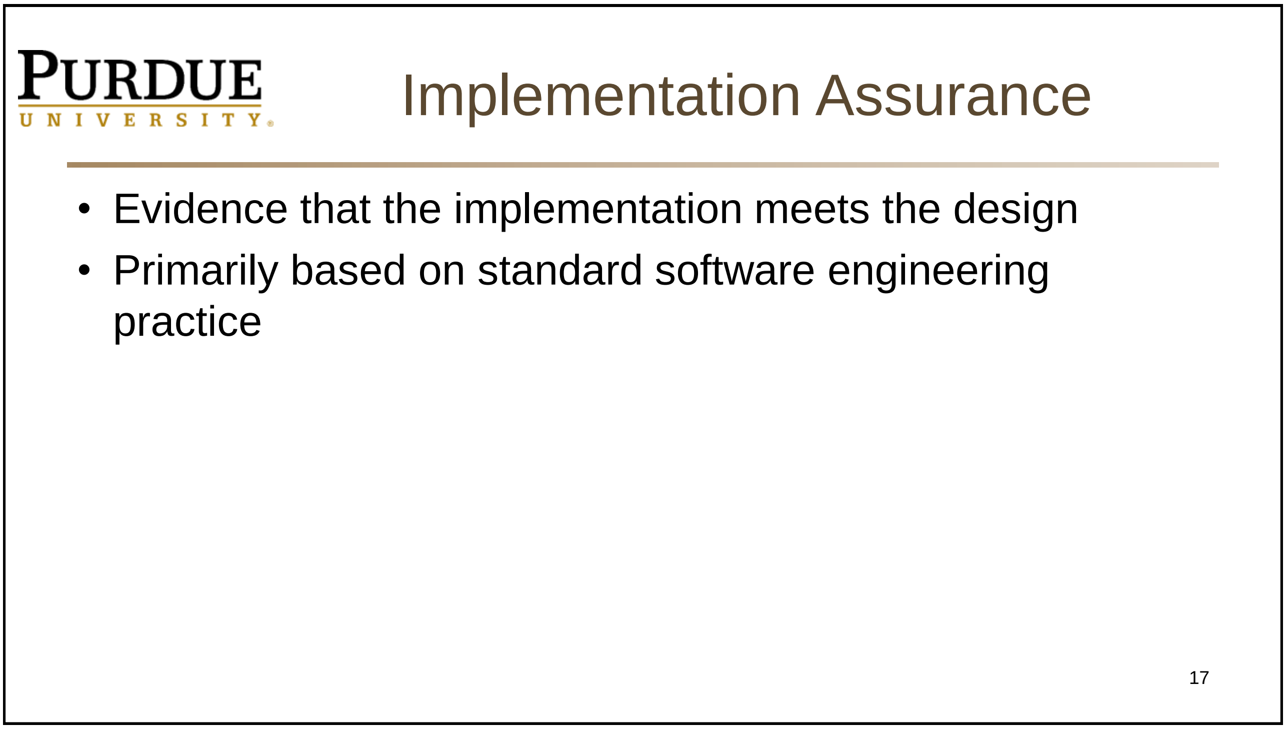
Requirements

- Statement of Goals that must be satisfied
- Security Policy is a requirement
- Security Model is a means of detecting/preventing errors, omissions in security policy
- Policy Assurance: Evidence that security policy is complete/consistent/sound
  - *Achieved through use of model*



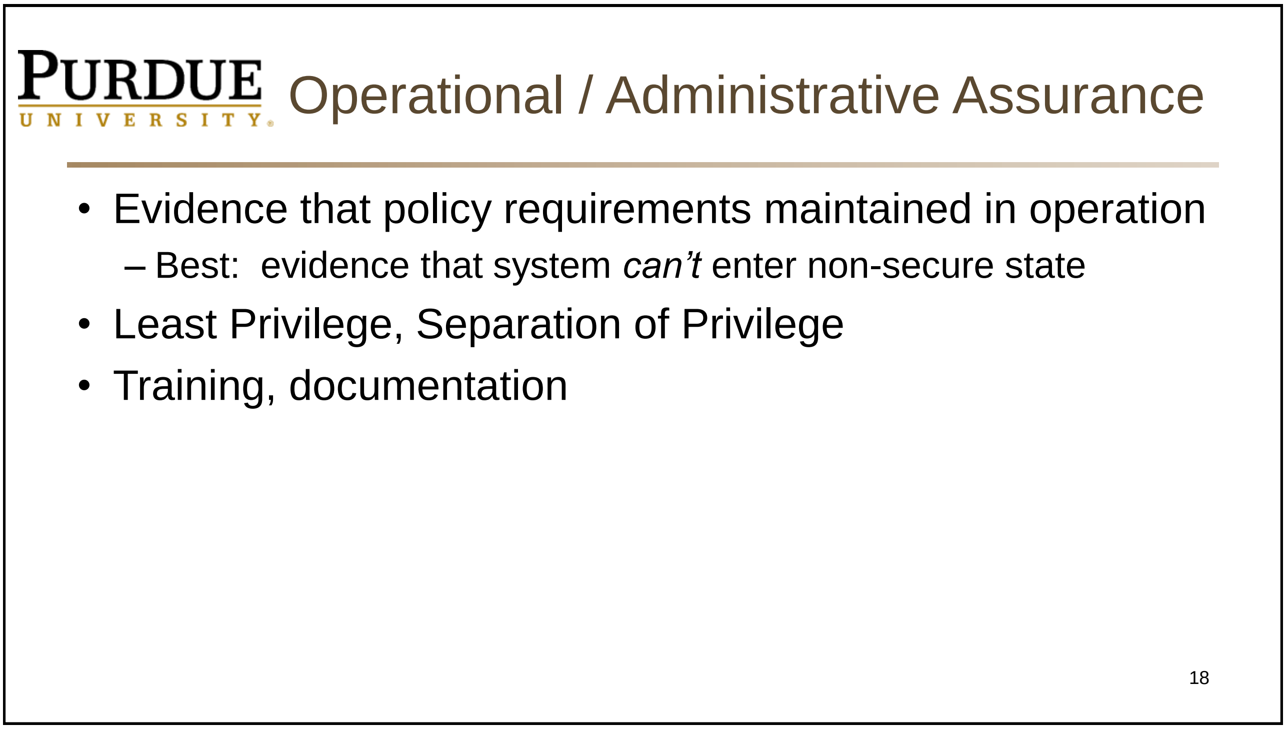
Design Assurance

- Evidence that Design meets Security Policy
  - Validation / verification techniques
  - We’ll discuss these later



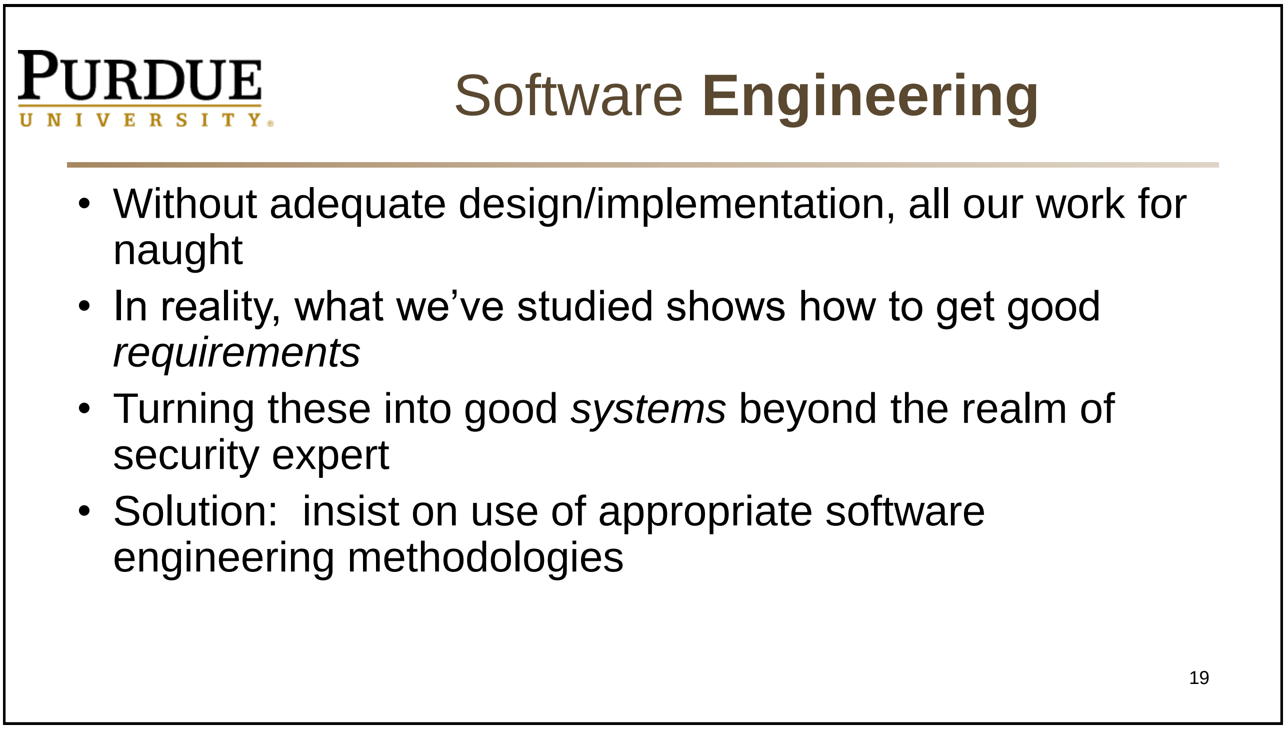
Implementation Assurance

- Evidence that the implementation meets the design
- Primarily based on standard software engineering practice



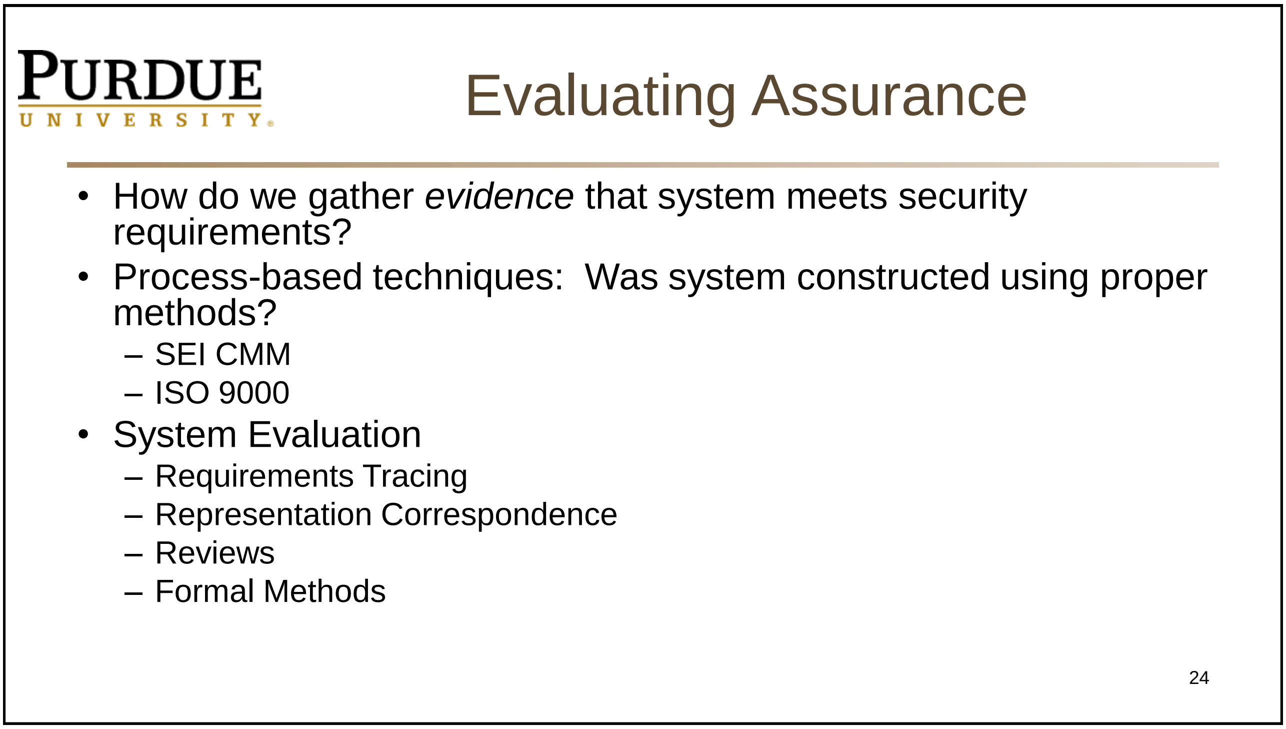
Operational / Administrative Assurance

- Evidence that policy requirements maintained in operation
  - Best: evidence that system *can't* enter non-secure state
- Least Privilege, Separation of Privilege
- Training, documentation



Software **Engineering**

- Without adequate design/implementation, all our work for naught
- In reality, what we've studied shows how to get good *requirements*
- Turning these into good *systems* beyond the realm of security expert
- Solution: insist on use of appropriate software engineering methodologies



Evaluating Assurance

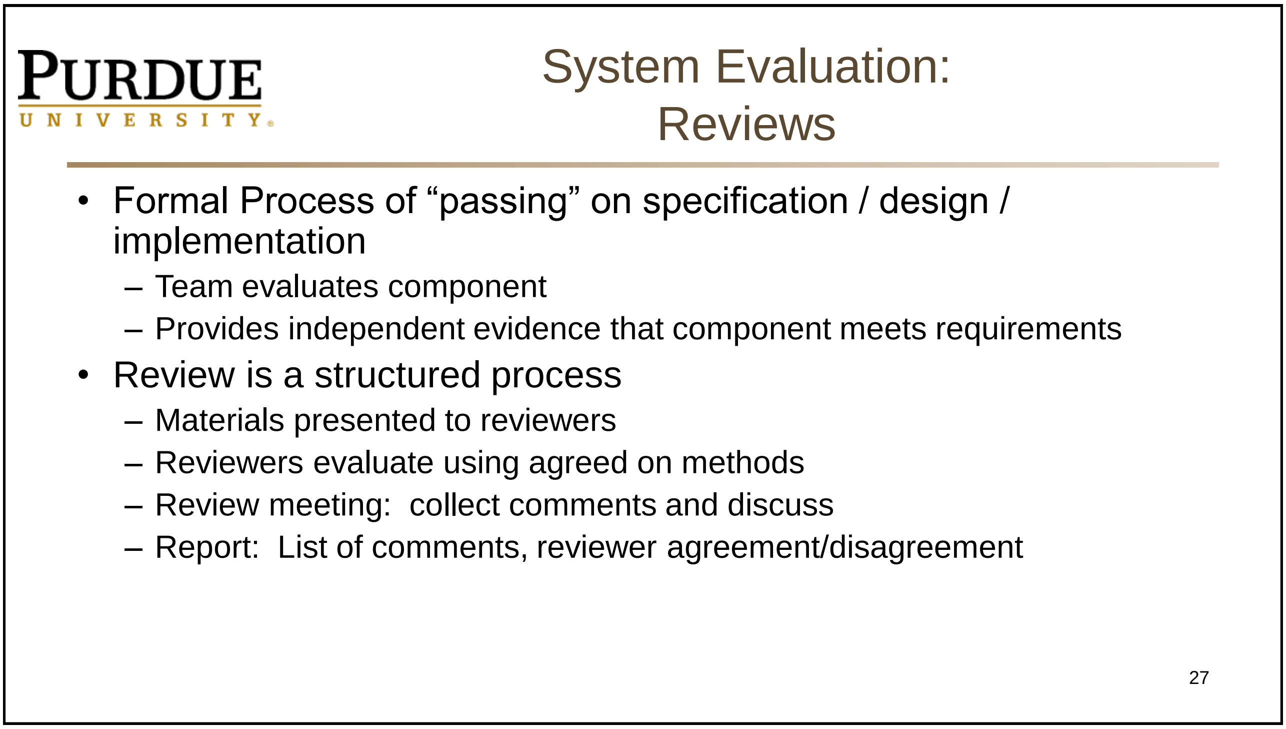
- How do we gather *evidence* that system meets security requirements?
- Process-based techniques: Was system constructed using proper methods?
  - SEI CMM
  - ISO 9000
- System Evaluation
  - Requirements Tracing
  - Representation Correspondence
  - Reviews
  - Formal Methods

Process Based Techniques

- Software Engineering Institute Capability Maturity Model (SEI CMM)
  - Specifies levels of process maturity
  - Criteria to evaluate level of an organization
- ISO 900[0-?] similar
  - More directed to manufacturing than software
- Configuration Management
  - Log/track changes
  - Ensure process followed
  - Regression testing / update, release control

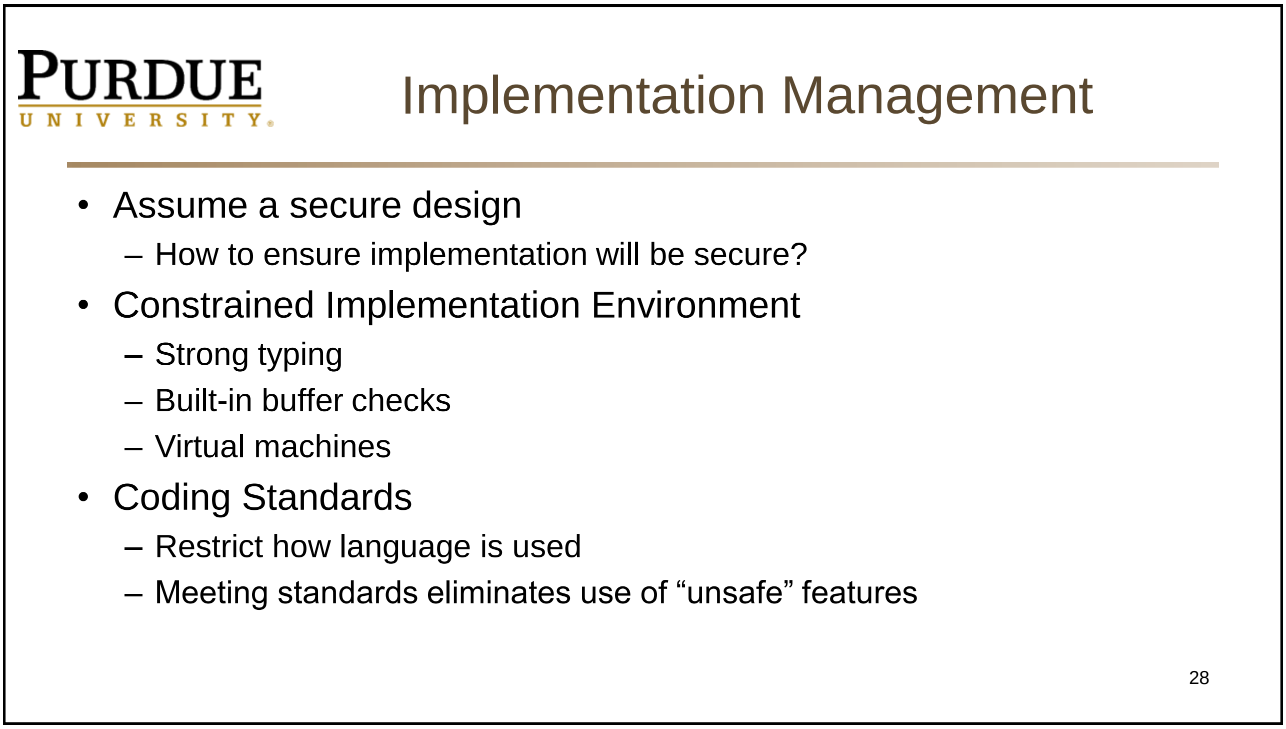
System Evaluation

- Requirements Tracing
  - Track requirement to mechanism
  - Ensures nothing forgotten
  - *Doesn't ensure it is correct*
- Representation Correspondence
  - Requirements tracing between levels
- Validating Correctness:
  - Informal arguments
  - Formal verification
    - May use automated tools



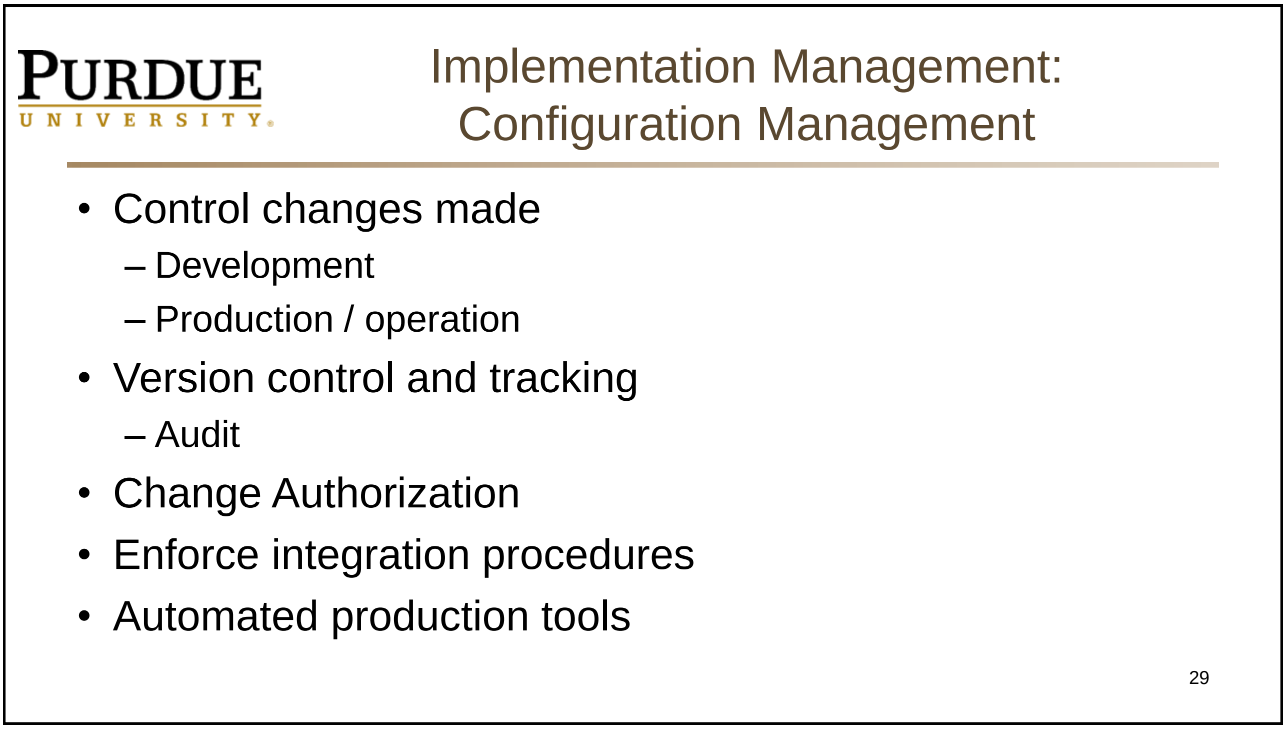
System Evaluation: Reviews

- Formal Process of “passing” on specification / design / implementation
  - Team evaluates component
  - Provides independent evidence that component meets requirements
- Review is a structured process
  - Materials presented to reviewers
  - Reviewers evaluate using agreed on methods
  - Review meeting: collect comments and discuss
  - Report: List of comments, reviewer agreement/disagreement



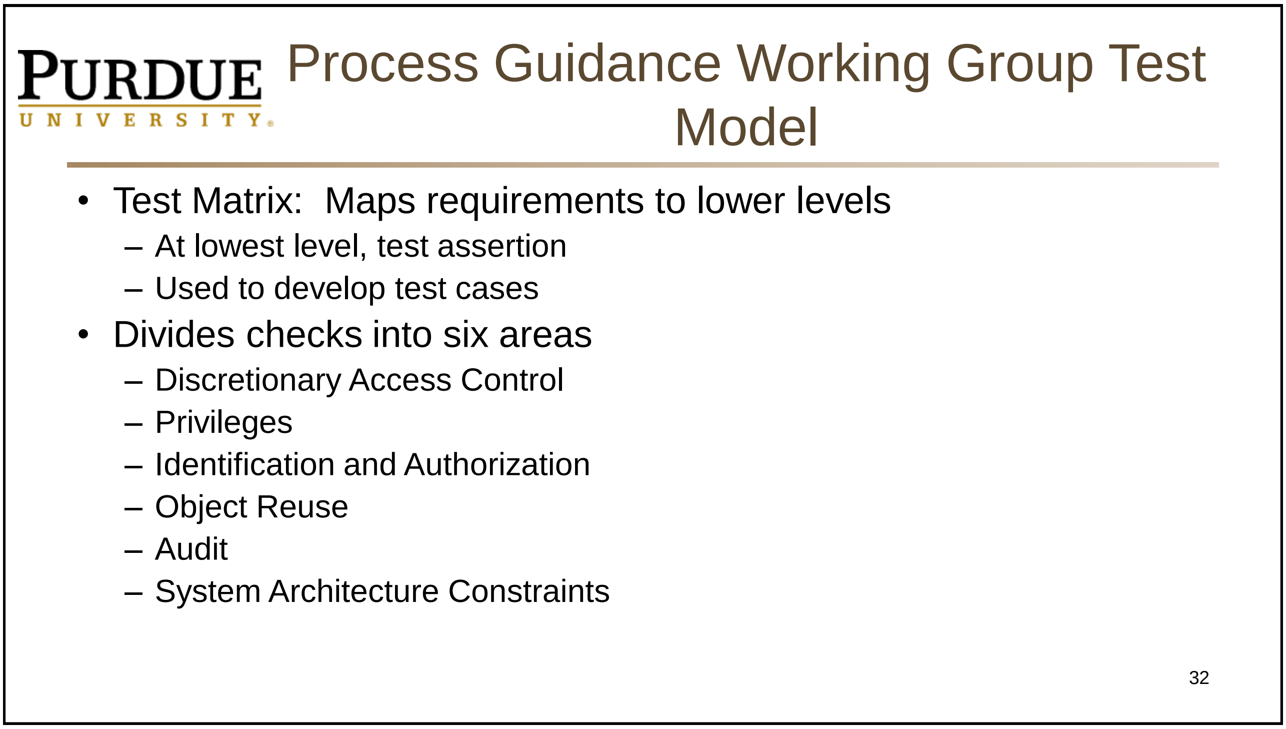
Implementation Management

- Assume a secure design
  - How to ensure implementation will be secure?
- Constrained Implementation Environment
  - Strong typing
  - Built-in buffer checks
  - Virtual machines
- Coding Standards
  - Restrict how language is used
  - Meeting standards eliminates use of “unsafe” features



Implementation Management: Configuration Management

- Control changes made
  - Development
  - Production / operation
- Version control and tracking
  - Audit
- Change Authorization
- Enforce integration procedures
- Automated production tools



Process Guidance Working Group Test Model

- Test Matrix: Maps requirements to lower levels
  - At lowest level, test assertion
  - Used to develop test cases
- Divides checks into six areas
  - Discretionary Access Control
  - Privileges
  - Identification and Authorization
  - Object Reuse
  - Audit
  - System Architecture Constraints

PURDUE
UNIVERSITY

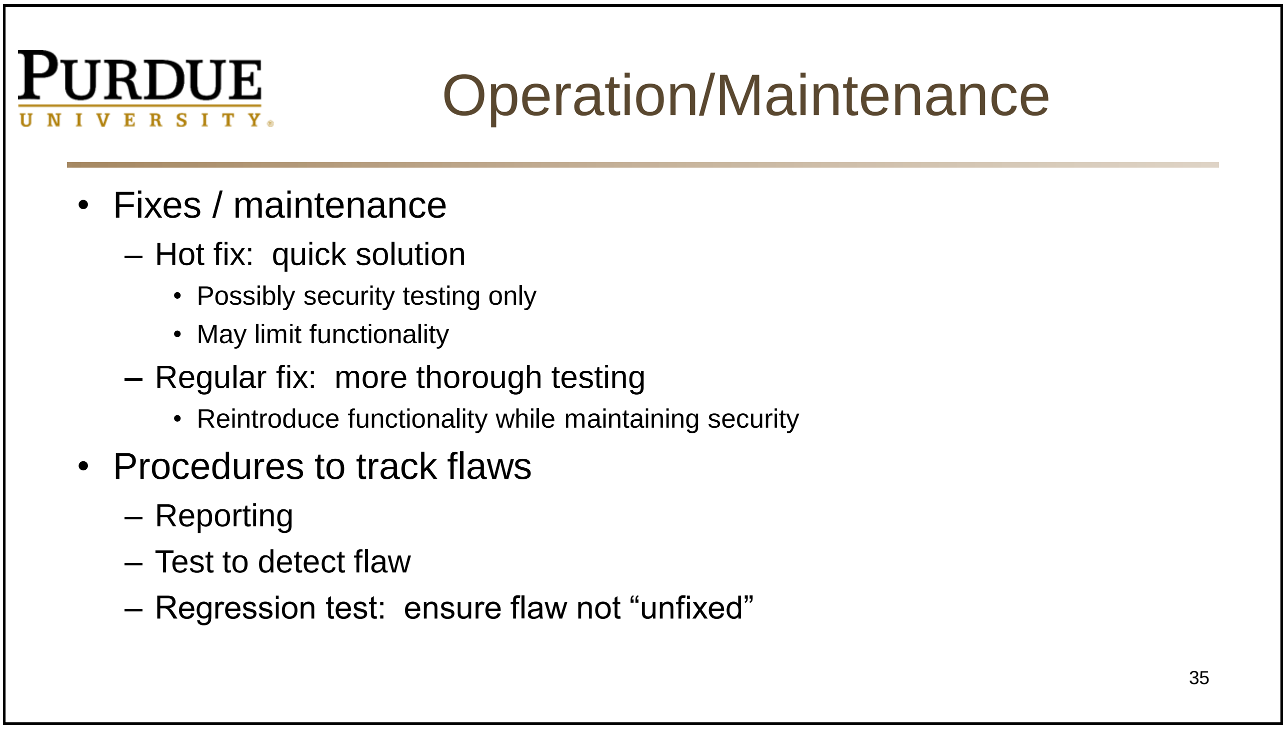
Top-Level Matrix: OS Example

Component	*DAC*	*PRIV*	*I&A*	*OR*	*Audit*	*Arch*
Process Management					✓	
Process Control	✓	✓		✓	✓	✓
File Management	✓	✓		✓	✓	✓
Audit		✓	✓	✓	✓	✓
I/O interfaces	✓	✓	✓	✓	✓	
I/O device drivers		✓		✓	✓	✓
IPC management	✓	✓		✓	✓	✓
Memory management	✓	✓		✓	✓	✓

PURDUE
UNIVERSITY

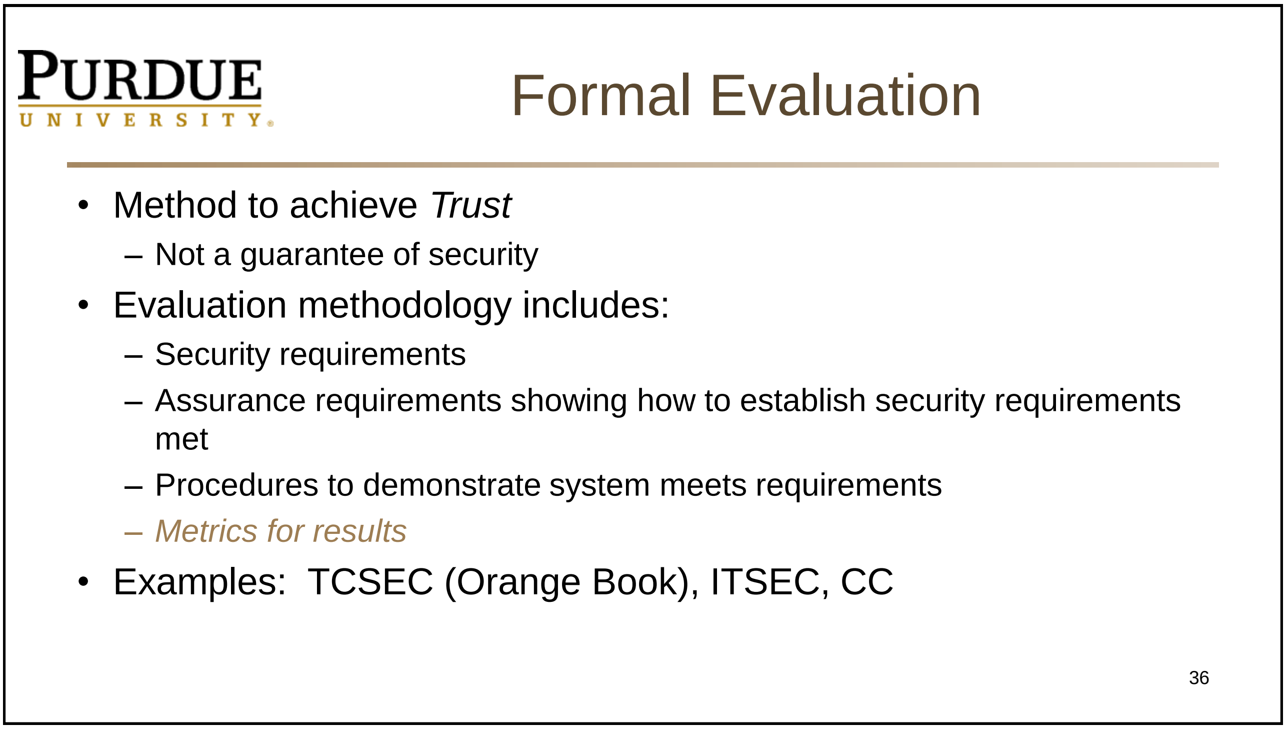
PGWG Test Model

- Each row generates lower level matrix
- Continue until test assertions possible
  - Verify only root can use *stime* successfully
  - Verify audit record generated for call to *stime*
- Develop test case specification for each assertion
  - Call *stime* as root: time should change, audit generated
  - Call *stime* as non-root: no change, fail, audit generated
- Develop test for each specification



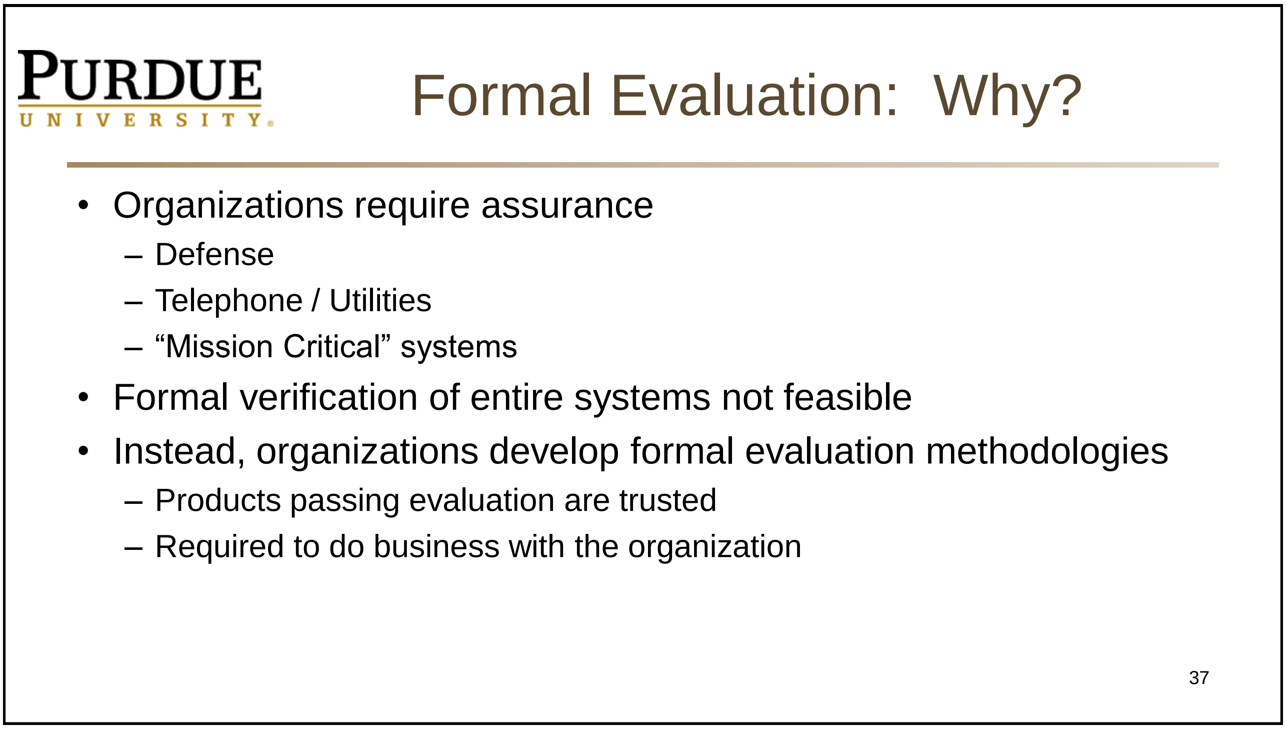
Operation/Maintenance

- Fixes / maintenance
  - Hot fix: quick solution
    - Possibly security testing only
    - May limit functionality
  - Regular fix: more thorough testing
    - Reintroduce functionality while maintaining security
- Procedures to track flaws
  - Reporting
  - Test to detect flaw
  - Regression test: ensure flaw not “unfixed”



Formal Evaluation

- Method to achieve *Trust*
  - Not a guarantee of security
- Evaluation methodology includes:
  - Security requirements
  - Assurance requirements showing how to establish security requirements met
  - Procedures to demonstrate system meets requirements
  - *Metrics for results*
- Examples: TCSEC (Orange Book), ITSEC, CC



Formal Evaluation: Why?

- Organizations require assurance
  - Defense
  - Telephone / Utilities
  - “Mission Critical” systems
- Formal verification of entire systems not feasible
- Instead, organizations develop formal evaluation methodologies
  - Products passing evaluation are trusted
  - Required to do business with the organization