



ILLINOIS INSTITUTE  
OF TECHNOLOGY

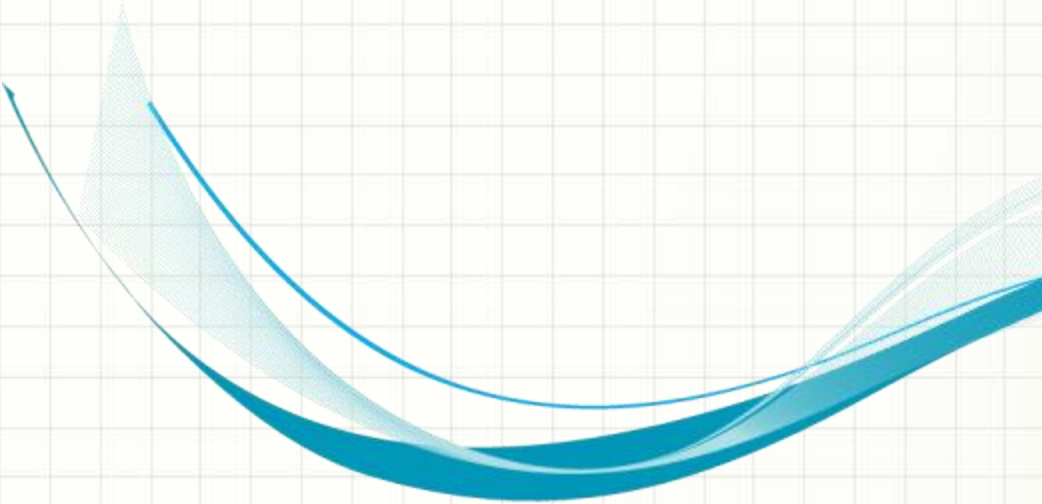
*Transforming Lives. Inventing the Future.*

[www.iit.edu](http://www.iit.edu)

# SOFTWARE ENGINEERING

## CS 487

Prof. Dennis Hood  
Computer Science



# Lecture 6

## Dependability and Reliability

# Lesson Overview

- Dependability and Reliability
- Reading
  - Ch. 9 – Software Evolution
  - Ch. 10 – Dependable Systems
  - Ch. 11 – Reliability Engineering
  - Ch. 12 – Safety-Critical Systems
- Objectives
  - Understand the concepts of dependability and reliability in the context of systems
  - Examine approaches for developing better systems, where better means dependable and reliable
  - Discuss assessment approaches for measuring these factors
  - Discuss system reengineering as a means for gaining insight into the design of legacy systems

# Participation – P5

1. Write a requirement stating how reliable you want your smartphone to be.
2. Identify your smartphone's partners (w.r.t. reliability).
3. Describe the interface between your smartphone and each of its partners and describe the protocol needed to support each interface.
4. Assess the risk exposure associated with your smartphone failing to achieve the reliability stated in #1, and explain the dependency on the partners.
5. Describe an exception which would prevent your smartphone from providing you with acceptable service. How could your smartphone detect and handle the exception such that adequate service is still provided?

# Dependability Considerations

- Repairability
  - The ability to recover from failure
  - Diagnosis, analysis, “surgical” repair, etc.
- Maintainability
  - Economical adaptation to new requirements
- Survivability
  - The ability to withstand “attack”
  - Recognize, resist, and recover
- Error tolerance
  - Avoid or at least tolerate user errors
  - Autocorrect if possible
  - Teach the user along the way



# Specification

- Types of specification
  - Risk-driven – avoid hazards
  - Reliability – measurable performance standards
  - Security – authorization and protection
- Formal specification
  - Human communication is complex and error prone
  - Formality seeks to simplify and reduce the opportunity for error
  - Unfortunately formality has had limited effectiveness in practice to date

# Risk Management

- Assessment
  - Identify assets requiring protection and value
  - Identify threats and likelihood of occurrence
  - Assess exposure (likelihood X impact)
  - Consider mitigation possibilities and costs
  - Mitigate where feasible
- Life-cycle risk assessment
  - Secondary assessment following system and data architecture decisions

# Failure Categories

- Hardware failure
  - Design errors
  - Component failure
- Software failure
  - Requirements issues
  - Design errors
  - Coding defects
- Operational failure
  - User misuse



# Safety Critical Systems

- Primary safety-critical systems
  - Embedded system controllers
  - Failure of the controller leads to failure of the system it is controlling
- Secondary safety-critical systems
  - Failure of this system will not directly cause harm
  - However such a failure could lead to harmful situations (e.g., CAD or CASE tools)

# The Need for Security

- Openness has many benefits
  - Data sharing
  - Remote user access, etc.
- But also introduces vulnerabilities
  - Unauthorized access
  - Denial of service
  - Exposure of sensitive data, etc.
- Security engineering attempts to develop systems that minimize the exposure
  - Application security is a software engineering problem
  - Infrastructure security is a systems engineering problem

# Security Management

- Access
  - User and permission management
  - Restrict users' access
- Deployment
  - Control installation and configuration
  - Patching
- Attacks
  - Monitoring, detection and recovery

# Security Concepts

- Asset – system resource that must be protected
- Exposure – potential loss/harm
- Vulnerability – exploitable weakness
- Attack – exploitation of vulnerability
- Threats – circumstances under which attacks can occur
- Control – a protective measure that reduces vulnerability

# Design for Security

- Architectural design
  - Protect critical assets
  - Distribute assets to minimize the effects of an attack
  - Analogous to a medieval castle
- Design guidelines
  - Establish and adhere to policies
  - Minimize impact through distribution, redundancy, compartmentalization, etc.
  - Recoverability, failing securely, safe deployment
  - Maintain usability
  - Validate inputs

# Design for Deployment

- Include support for viewing and analyzing configurations
- Minimize default privileges to only those that are essential
- Localize configuration settings
- Make it easy to fix vulnerabilities



# Dependable Programming

- Control the visibility of information
- Check all inputs for validity
- Handle all exceptions
- Avoid error-prone code constructs
- Provide recovery and restart capabilities
- Check array bounds
- Include timeouts when interfacing with external components
- Name all constants that represent real-world values

# Risky Programming

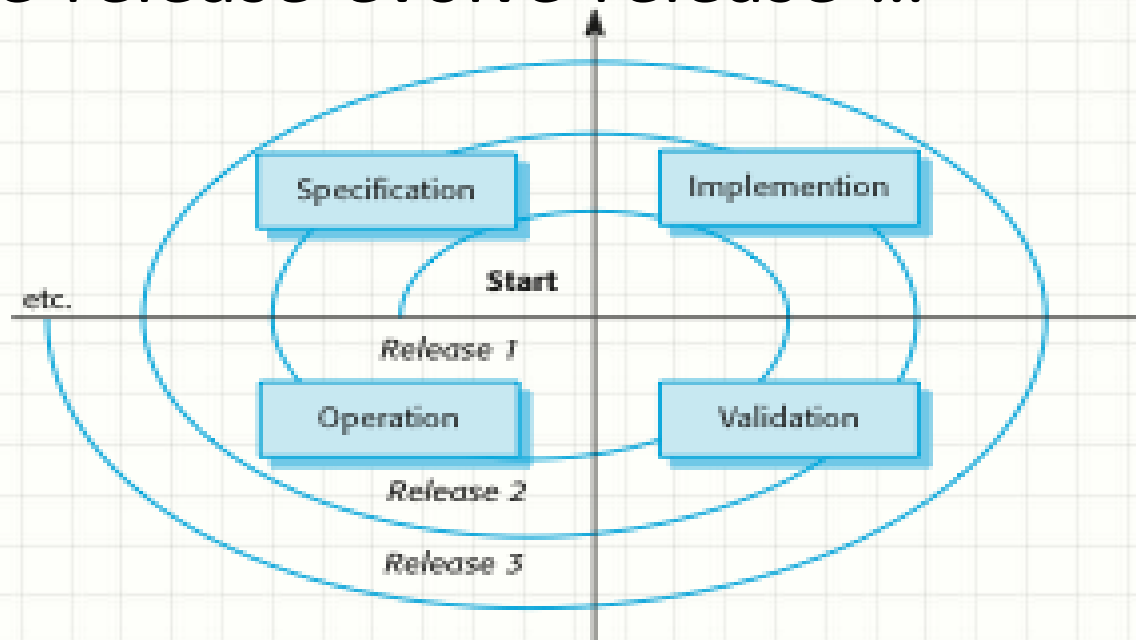
- Unconditional branching (go-to's)
- Floating point numbers
- Pointers
- Dynamic memory allocation
- Parallelism
- Recursion
- Interrupts
- Inheritance
- Aliasing
- Unbounded arrays
- Default input processing

# Survivability

- Design to minimize vulnerabilities and their effects,
- but just in case, design to withstand attacks
- This is critical for critical systems
- Multiple strategies
  - Resistance to attacks
  - Recognition of the type of attack
  - Maintain adequate operation during an attack and then recover to full operation

# Software Evolution

- Manage the inevitable change
  - Support and facilitate business growth
  - Take advantage of technology innovation
- Evolve-release-evolve-release-...



# Evolution Dynamics

- Grow or progressively lose value (e.g., user satisfaction, perceived quality, etc.)
- Evolution tends to increase complexity
- Bigger systems tend to resist evolution
- Organizational bureaucracy dampens evolution
- The bigger the evolution the greater the number of associated problems

# Managed Change

- Formal change requests/proposals
  - Purpose and priority
  - Cost and effort
  - Risk assessment
- Change review and authorization
  - Change control board
- Change implementation
  - Software engineering
  - Planning and management
- Change release
  - Communication
  - Rollback planning



# Program Evolution Dynamics

- Change is inevitable
  - As long as the system is used there will be demand to correct imperfections, improve shortcomings and add functionality
- Change increases complexity
  - Adding to an existing structure tends to degrade stability
  - New functionality may bring new defects
- Change tends to be regulated by factors such as system and organization size
  - Big systems are more difficult to change
  - Bureaucracies impede change

# Maintenance

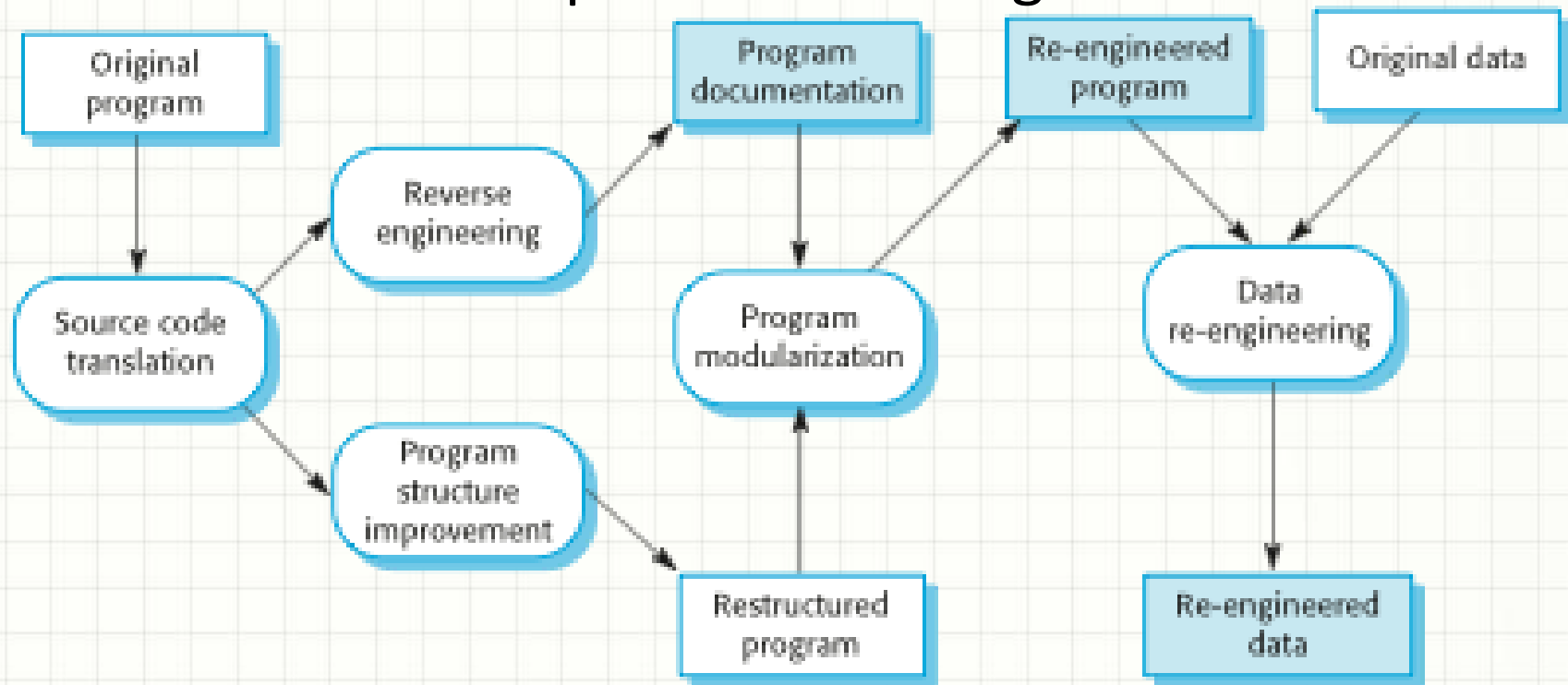
- Development effort and discipline should reduce maintenance effort and cost
  - Better analysis will result in a better alignment with user needs
  - Better design and implementation will reduce defects (including user confusion, etc.)
  - More thorough QA will minimize defects in production
- The evolutionary approach can result in higher maintenance costs
  - Adding functionality to an existing system is more difficult
  - This issue should be countered by planning evolutions well in advance where possible
- Change should be managed with discipline
  - Defect removal
  - Enhancements

# Drivers of Maintenance Activity

- Interfaces
  - Number and complexity matter
  - User and system interfaces
- Information
  - Number of data sources
  - Data structure complexity
- Volatile requirements
  - Policies and procedures
  - Business rules
  - Technology
- Processes utilizing the system
  - The more users, the more demand for change

# Reengineering to Gain Understanding

- Benefits
  - Creates a newer, more maintainable version
  - Faster and cheaper than building brand new



# Refactoring

- Preventive maintenance thru occasional touching up to stave off degradation
  - Improve structure and performance
  - Reduce complexity
  - Improve understandability, etc.
- Improve what's already there, don't add new functionality
- Targets for refactoring improvement
  - Removal of duplicate code
  - Decomposing long methods
  - Simplify or replace “switch” statements
  - Encapsulate recurring “clumps” of data
  - Remove speculative generality

# Legacy Systems

- Systems supporting critical business systems may hang around for a while
  - Change is risky
  - Need downtime to switch over
  - Domain knowledge seeps away over time
- Possible next steps
  - Scrap it
  - Leave it as is and maintain
  - Reengineer it to improve maintainability
  - Replace all or at least some of it
- Assess business value vs. system quality



# Legacy System Evaluation

