Software development is a leading cost for many institutions, whether these projects are for in-house use, contract development, or for product sales. In order for an institution to meet its mission goals, it is advantageous to have methodologies and tools that streamline the software development life cycle without sacrificing quality. Improving quality while gaining in efficiency, ensuring continuity with teams susceptible to employee turn-over, adapting to constantly evolving technologies, and pursuing new opportunities present major challenges in software development for highly scientific and engineering applications.
The Problem: Complexity

- Since the beginning of software development projects, there has been a persistent set of problems that result in time and cost over-runs. Delivered products can fail to fully meet project goals, can require substantial maintenance efforts, and can rapidly become obsolete. Over the decades, various methodologies have evolved to address these risks. For example, “top-down” or “waterfall” development paradigms have given way to an object-oriented (OO) Unified Modeling Language (UML) and iterative life cycle management. In addition, tools have evolved out of algorithmic programming languages and centralized mainframe operating systems into computer-aided OO programming languages and networked operating systems.
Project Development Risks

- **Constant evolution of computer technology** (both hardware and software) that outpaces project lifetimes; this results in heterogeneous computer systems, extensive staff retraining, subsystem incompatibilities, difficult configuration and control, and prematurely obsolete products;
- **Changes in project requirements**; accurate requirements specification at the beginning of projects is an exceedingly difficult process and this is particularly true in scientific and engineering disciplines underlying critical operational missions
- **Excessively complex database management systems (DBMS)**; a major problem has been the lack of suitable database management systems since commercial DBMS applications are designed for business, not science and engineering, and often fail to meet the unique needs of scientific and engineering projects.
A New Software Development Process

- Based on iterative “light and fast” development and testing. It addresses issues of:
  - Portability (device independence)
  - Extensibility (change in project scope)
  - Robustness (fail-safe modes)
- Emphasis is on the use of OO software with:
  - Inheritance (code reusability)
  - Encapsulation (simplifying software by hiding extraneous details)
  - Polymorphism (individual methods with a range of operations)
  - “Plug and play” software components

The development process is based on iteratively re-factoring (building and testing) of manageable software subsystems. At the early stages of a project, rapid-prototyping is employed to address specific risk areas before a detailed system is designed. At each iteration of development and testing, software components are subdivided or merged to address new problems revealed by testing. Small, individual software components have the capability to automatically self-validate themselves and are easily modified without effecting larger programs that include them. The development tools and the software and hardware architecture can be changed or upgraded at each iteration.
An Integrated Hardware and Software Architecture

- Leveraged advanced computer hardware and software applications and heterogeneous peer-to-peer (P2P) distributed computer systems; computer systems advance incrementally.
  - Redesigning an entire system using homogeneous, sole-source components frozen at specific versions is more expensive in the long run than incrementally upgrading individual components in a highly versatile and adaptable system.
- With a distributed, decentralized system, individual workstations employ peer-to-peer communications and multi-tiered design patterns and easily adapt to changes in single component.
Constructive Iterative Systems Development Lifecycle (CI-SDLC)

An improved methodology is based on the definition of two complementary concepts: an *Initial Framework Life Cycle* that streamlines the development process of a new system and a *Spiral Refactoring/Testing Life Cycle* that encapsulates the iterative, retesting and retooling processes. The two concepts are phases of a unique systems engineering process called the *Constructive Iterative Systems Development Lifecycle (CI-SDLC)*, developed by SET based on its prior experience of transitioning models into operations, and which are essential for its implementation.
The Spiral Re-factoring/Testing Life Cycle (RTLC) is an iterative life cycle that operates on the initially designed prototype system. It immediately follows the original design and implementation and starts with a planning phase that estimates the project time and costs for one complete iteration. In the retool/prototyping phase new computers and software applications can replace the old ones to create an updated prototype.
UML View of a Database

- Scientific and engineering data records are fundamentally different from the financial data records typically used in the business community.
- Commercial databases tend to be relational while scientific and engineering data bases tend to be hierarchical.
- Sometimes a small project team only needs a good file management system that can later be scaled for instrumental or archival applications.
- Standardized common database format such as CDF, net-CDF, HDF, and FITS are often difficult to implement across disciplines and stakeholders.
- Abstract objects can be used to encapsulate the specifics of a data record or file.
# Software Development Life Cycle Philosophy

1. Emphasizes the development of prototyping solutions in critical risk areas prior to a major project development
2. Requires unit components to include self-validating testing
3. Includes the concept of “graceful failure”, i.e., the system adjusts to component failures by substituting alternate estimates of data
4. Employs advanced and proven COTS products that enhance the system design and development process, e.g., UML tools
5. Encapsulates data, core models, and legacy software in a device-independent OO programming language, e.g., JAVA
6. Streamlines the process of moving from a concept of operations through unit development to integration testing and delivery
7. Considers “the human factors” inherent in successful development
8. Supports joint application development by project teams from separate institutions
9. Leverages advanced computer hardware and software
10. Scales the database applications to the project needs
Best Engineering Practices

• Employ methods consistent with the Capability Maturity Model (CMM)
  – Project planning and management (strategic goals, costs, schedules)
  – Process (life cycle) management and software configuration control
  – Object-oriented analysis, design, and implementation using a Unified Modeling Language (UML)
  – Object-Oriented system design and development
  – Device-independent software applications
  – Rapid prototyping: the unit testing of key risk components prior to system design, commercial of-the-shelf software (COTS)
  – Iterative requirement/design and development/testing lifecycle
  – Integrated validation and verification
  – Training, Maintenance