How eRA Develops Software

Executive Summary: In an organized, iterative, carefully documented manner, eRA has built a dense infrastructure of methodology, standards, and procedures to underpin the development of a reliable enterprise system for electronic research administration.

For software development and quality assurance, NIH has relied thus far solely on contractors. Now the retirement of the legacy IMPAC system is freeing up NIH programmers to join the development team.

eRA's methodology combines the classical Waterfall method and the prototype-oriented Spiral method. It proceeds in three phases:

1. Definition Phase: systems analysis and software development analysis;
2. Development Phase: software design, code generation, and testing; and
3. Maintenance Phase: perfective maintenance and other kinds of maintenance, documentation, and training.

To capture the needs of users at every step, eRA employs focus groups, user groups, and the group advocates. Key features of the process are the use of database and tools technology from Oracle and reliance on a strongly modular approach that permits systematic upgrading of modules on a regular basis. **End Summary**

*****

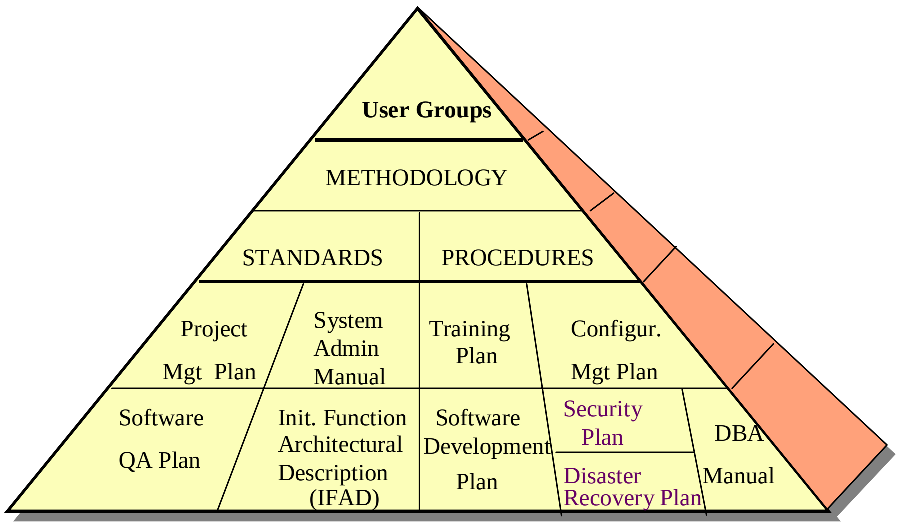
For eRA, developing software is a mission-critical business. With some 576,000,000 transactions taking place annually in IMPAC II and Commons, the functioning of NIH's extramural grant and contract system relies on a dependable and effective software development process.

eRA has built a dense project infrastructure (see figure) to ensure the overall integrity of the system. Other features of the project also contribute:

- for database technology, we rely on Oracle, a leading vendor with robust products;
- our strongly modular approach has enabled us better to target the needs of particular user groups while reducing the complexity of each piece of the project; and
- we have moved ever farther toward empowering users and involving them at every step. Our recent Prioritization Exercise brought forth a highly articulated set of user needs, with estimated hours and required funds. Now we are working on methods whereby, through their group advocates, user groups can play a key role in bug-fixing and ongoing requirement prioritization.

To ensure development that is both rapid (i.e., as efficient as possible) and high-quality, eRA employs a blend of two major models of software development: the classic Waterfall method and the Spiral method first advocated by Barry Boehm.

eRA PROJECT INFRASTRUCTURE



The classic Waterfall method begins with target system requirements and progresses through analysis, design, development, acceptance, installation, and maintenance. This approach rates high in terms of logical planning, development, and solidity; but its inflexibility to changes in user requirements can lead to completed systems that must undergo expensive revisions. So eRA balances this approach with one that relies more on prototyping—the so-called Spiral method.

eRA’s prototypes typically use disposible code to demonstrate concepts, process flows, and capability. In an iterative, phased way, we gradually build out one element after another in a manner that reduces risk and permits flexible adaptation to evolving user needs. Users are deeply involved in prototype development.

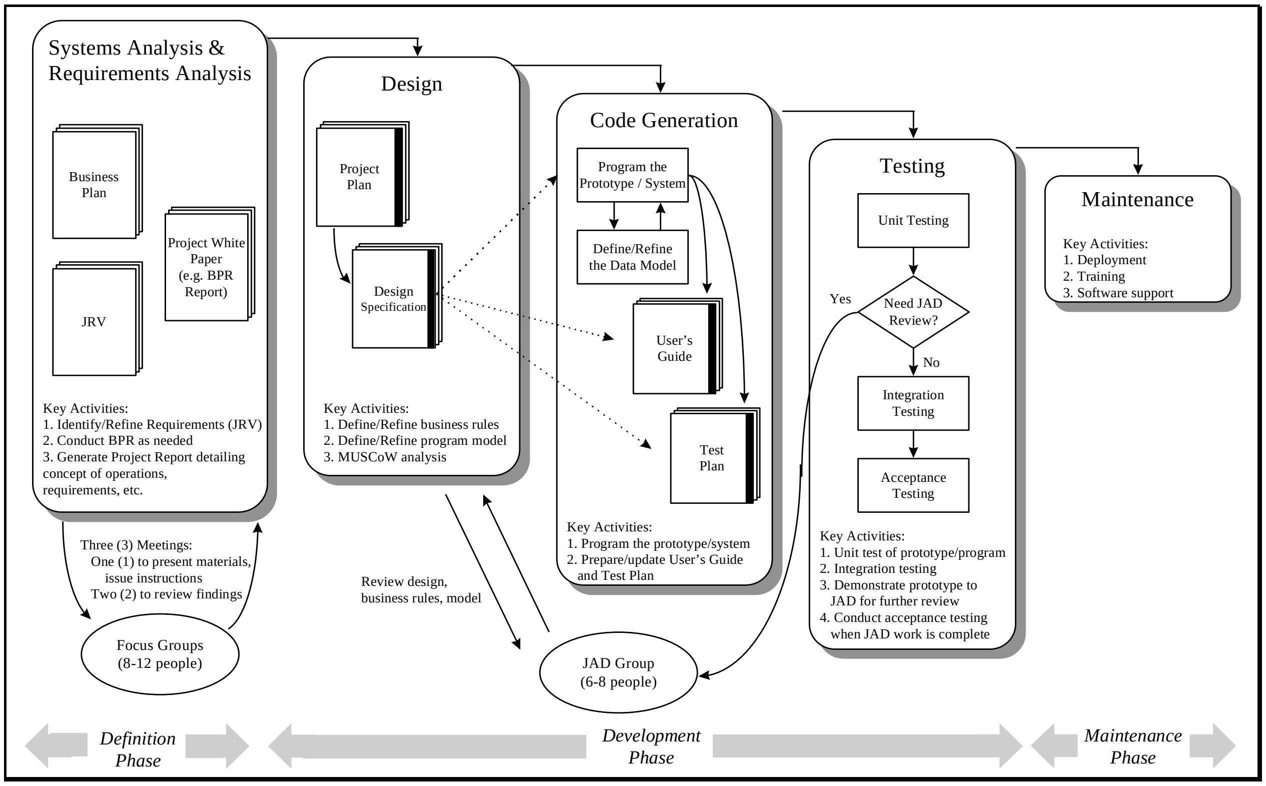
eRA favors the Waterfall method in the Definition Phase (systems analysis and requirements analysis) as well as in the Maintenance Phase (correcting defects, making improvements, and supporting users). The Development Phase, in contrast, tends to follow the Spiral method.

Definition Phase

In the Definition Phase eRA specifies the scope, functions, and capabilities of the resultant system by identifying the requirements of the system and software. Specifically, we define the data, functions, performance guidelines, interfaces, design constraints, and validation criteria for the completed system.

In the Definition Phase, an eRA Task Team from NIH and its contractors first undertakes Systems Analysis to define the role and scope of the software system in the context of general planning.

Software Requirements Analysis


Systems Analysis & Requirements Analysis
Business Plan
Project White Paper (e.g. BPR Report)
JRV
Key Activities:
1. Identify/Refine Requirements (JRV)
2. Conduct BPR as needed
3. Generate Project Report detailing concept of operations, requirements, etc.
Three (3) Meetings:
One (1) to present materials, issue instructions
Two (2) to review findings
Focus Groups (8-12 people)
Design
Project Plan
Design Specification
Key Activities:
1. Define/Refine business rules
2. Define/Refine program model
3. MUSCoW analysis
Review design, business rules, model
JAD Group (6-8 people)
Code Generation
Program the Prototype / System
Define/Refine the Data Model
User's Guide
Test Plan
Key Activities:
1. Program the prototype/system
2. Prepare/update User's Guide and Test Plan
Testing
Unit Testing
Need JAD Review?
Yes
No
Integration Testing
Acceptance Testing
Key Activities:
1. Unit test of prototype/program
2. Integration testing
3. Demonstrate prototype to JAD for further review
4. Conduct acceptance testing when JAD work is complete
Maintenance
Key Activities:
1. Deployment
2. Training
3. Software support
Definition Phase
Development Phase
Maintenance Phase

Requirements analysis asks questions about the function, capability, and performance of the new software system; the constraints that will limit its development and operation; and the criteria that will be used to validate it. It seeks to define what role the software will play in the business system and what features are needed to fit this role.

The Task Team reviews existing eRA and assignment application systems, meeting with users and gathering information from the group advocates as representatives of the eRA user group process at NIH. The Task Team defines the project scope, process flow, and many of the requirements, formulating the results in an initial draft of a *Module Initiative Report*. The Task Team then meets with a select group of representatives from different NIH Institutes and Centers (ICs)—or, for Commons, with representatives of the research community—in two focus groups that prepare comments about the processes and requirements in the report.

Development Phase

The Development Phase contains three steps: Software Design, Code Generation, and Testing.

Software Design

In the *Software Design* step, the requirements and capabilities collected and approved during the Definition Phase are translated into instructions defining the architecture, appearance, flow, and operation of the software system. The development team prepares a series of prototypes that iteratively define and refine the requirements of the process. These prototypes, along with the project documentation, are presented in joint application development (JAD) sessions to a select group of users whose input, comment, and critique help refine the planning and designing documents.

To start the Development Phase and the first step in software design, the design leader prepares draft copies of the *Project Plan* (defining scope) and *Design Specification* (defining development efforts) for use in the JAD sessions. JAD members focus their efforts on establishing priorities for requirements/features, developing phases for the software releases, and ensuring all requirements are uncovered. This is the time for MuSCoW (Must/Should/Can/Wish) analysis that establishes the relative essentiality of users' needs.

The *Design Specification* describes the system architecture by using the system requirements to develop a data model and the requirements to develop the software model. Because software application systems invariably require modification after implementation, it is extremely important to build a software model that provides clear distinctions between the storage/use of the underlying data and the operation of the software application system.

Technical reviews are conducted to ensure the quality of the design and the accuracy of the design direction. Since the prototypes only provide initial representations of possible eRA application systems, reviews of the *Design Specification* are initially limited to Internal Design Review (IDR), performed by members of an eRA Task Team, including members of the Quality Assurance team. During the IDR, the design is reviewed for adherence to the software requirements and anticipated functions of the system. Deficiencies and changes identified by the IDR committee result in refinements of the design. The review of the prototype by the JAD group forms part of the Critical Design Review (CDR), conducted at the completion of the prototype phase.

Code Generation

Code Generation is a process of translating design instructions into programming language statements that can be automatically transformed into computer-executable instructions. The *Design Specification* contains the instructions that the development team needs to build the software application system and the accompanying database from the software model and data model, respectively. The data model defines the architecture for the database by identifying the data tables as well as their relationships with each other and with the software model. From the data model, detailed procedural instructions (e.g., SQL for a relational database management system such as eRA's) are constructed to build the tables, their indexes, and any accompanying constraints. The software model illustrates the layout of eRA application system screens, the relationship between the screens, and the instructions that must be performed.

The functional hierarchy defined in the design provides the structure for the software model and thus the modules for the resultant source code. The main module (top or parent) is coded first, then the functionality of the system is developed downward from the parent module, with increasing specificity. The functionality of the system can thereby be quickly coded and established while the details are developed during subsequent stages.

Technical reviews are conducted periodically to validate the source code. Code reviews are performed by "independent" team members--staff who did not develop the code in question. The focus of a review is to ensure that the code adheres to the instructions provided in the design document, conforms to coding standards, satisfies the software requirements, and makes use of common modules when possible.

Next the development team proceeds with creating an initial system prototype that displays the functionality of an eRA application system at the highest level. The JAD meets to respond to this prototype as well as the *Project Plan* and *Design Specification* documents, seeking to refine the system, verify its functionality and accuracy while identifying deficiencies and inconsistencies.

Testing/Quality Assurance

eRA takes a proactive approach to software quality by striving to avert problems and potential compromises in the design stage rather than resolving them later. Quality assurance staff participate in the technical reviews at both design and coding stages. This makes them aware of design decisions, the proposed functionality of the system, and the operation of the software so that they can be knowledgeable testers.

The *Design Specification* includes validation criteria for each module. These provide the foundation for constructing the *Software Test Plan*, written by the quality assurance staff. The validation of the source code and the application system against a checklist occurs thus:

1. **Unit Testing.** The contractor performs unit testing during prototyping and code generation to verify the software against the validation criteria listed in the *Design Specification*. Full integration testing is conducted only at strategic points;
2. **Integration Testing.** After the source code has been unit tested and accepted, it is incorporated into the entire application system and tested by a quality assurance contractor to discover problems arising from conflicts between the new module and existing modules.
3. **Acceptance Testing.** When the software is determined to be acceptable by the developing contractor, then acceptance testing begins. Acceptance testing is the most crucial testing activity in terms of verifying the client's acceptance of the software application system. Acceptance testing is completed when NIH accepts that the software application system satisfies the criteria defined in the *Software Test Plan*.

Maintenance Phase

Realistically, changes to software are inevitable. Expected maintenance activities include:

- **Corrective Maintenance:** efforts to identify the deficiencies, develop solutions, and implement them. eRA is revising its method of registering defect reports to give user group advocates a key role in the process.

- **Perfective Maintenance:** fulfilling new user requirements by following the development path from the design step, through code generation, testing, and into implementation.

- **Preventive Maintenance:** support staff identify ways to improve the system's efficiency, reliability, and flexibility. This may include reorganizing the data objects based upon changing usages, adding indexes to improve performance, and moving objects to reduce storage requirements.

- **Operational Maintenance:** reducing storage space and improve performance—for instance, rebuilding fragmented tables and their indexes to recover lost disk space and reduce query times.

- **Adaptive and Structural Maintenance:** updating software and hardware, including new devices such as optical drives.

Documentation and Training

What is termed the Maintenance Phase from the perspective of software development is clearly also the Functional Phase from the perspective of users. Essential to the success of a software system are documentation and training. eRA’s contractor develops and delivers documentation, including User’s Guides, to NIH as part of its contract. OER also produces documentation as well as providing training for users. eRA is currently undertaking a concerted effort to develop more user-friendly documentation to meet users’ needs to learn essential functions of the various modules as well as to identify additional valuable functionality in the system that can match their needs.