

JackBe's Application Methodology

Jerrold Prothero, Ph. D.
Usability Specialist and
Project Leader, JackBe
jerrold.prothero@jackbe.com

JackBe
June 2007



Table of Contents

Introduction	3
Principles and Procedures.....	4
2.1 Implementation Principles	4
2.2 Implementation Procedures.....	5
The JackBe Application Methodology.....	6
3.1 Overview	6
3.2 Inception Phase	7
3.2 Elaboration Phase.....	9
3.4 Construction Phase.....	12
3.5 Transition Phase.....	14
Appendix: Table of Artifacts	16

Index of Tables

Table 1: Inception Phase Activities	8
Table 2: Elaboration Phase Activities	11
Table 3: Construction Phase Activities.....	13
Table 4: Transition Phase Activities	14
Table 5: Project Artifacts	16

JackBe Application Methodology Whitepaper

Introduction

The JackBe Application Methodology (JAM) is an approach to software development that combines the benefits of User-Centered Design (UCD) with a light-weight version of the iterative Rational Unified Process (RUP).^{1 2 3 4}

UCD “builds the customer into the product” by systematically making sure that real end-user needs drive the development process. RUP is a mature, tested development methodology that provides a structured set of milestones and checkpoints to keep projects on course.

By explicitly integrating UCD with RUP, the goal is to ensure both that the right product is being built, and that it will be delivered successfully. JAM facilitates achieving the following critical objectives:

1. Identify and meet real market needs. This requires receiving and acting on market feedback throughout development.
2. Usability. The delivered software must be easy to learn and to use.
3. Robustness. The delivered software must be free of major defects and be easily maintainable.
4. Efficient implementation. Minimal calendar time and overall resources required.
5. Build confidence. Provide reassurance to stakeholders that the development process is under control.

Section 2 describes the core JAM principles. If for a particular project a decision has been made not to use RUP, these core principles are still applicable to almost any development methodology.

Section 3 describes the integration of UCD with RUP.

¹ http://en.wikipedia.org/wiki/Rational_Unified_Process

² <http://es.wikipedia.org/wiki/RUP>

³ <http://www.objectmentor.com/resources/articles/RUPvsXP.pdf>


⁴ For a high-level description of how to apply user-centered design principles to the XP and waterfall methodologies, see the GUIDe process model: http://www.interacta.fi/guide_en.html

Principles and Procedures

2.1 Implementation Principles

Jam is based on four general principles: user-centered design; iterative development; continual testing; and parallel development.

1. *User-centered design (UCD)*. (UCD) refers to building a software product from the perspective of the end-user. UCD calls for a “top down” analysis of user needs, followed by implementation of the user interface to meet these user needs, followed by building the functionality to support the user interface. Traditionally, software was designed “bottom up”, starting with functional requirements (what the software will do), implementing the most basic functional units, then integrating the functional units and in the process creating the user interface. UCD’s top-down approach has three main advantages over the traditional bottom-up approach. First, by building the interface at the start UCD provides an excellent communication tool between developers, management and end-users, to make sure that the right functionality is being implemented and that it is being presented in the best possible way. Second, UCD reduces the traditional integration nightmare of the bottom-up approach, since the software is integrated from the beginning. Third, it is a significant morale boost for all involved to see from very early on what the final product will look like. In part, the current trend towards UCD in the software development industry reflects the growing power of computer hardware. When computer hardware was the limiting problem, it made sense to start by analyzing software components, to make sure it could be implemented on available hardware. Now that supporting the user is the limiting problem, it makes sense to start from the user’s perspective.
2. *Iterative development*. Iterative development calls for building a system in a series of iterations, checking that the development is on course after each iteration. By contrast, the traditional “waterfall” software development model is one big process: it begins by defining all software requirements, then designs the software architecture, implements the software, tests and delivers. The waterfall model is the most efficient development approach for mature application areas where there is a clear understanding of how a particular type of software should be built. The iterative approach is better suited to the JackBe situation of a new kind of software technology and diverse application areas, where there is likely to be a need for efficient mid-course corrections during development.
3. *Continual testing*. The key to building success into software projects is continual testing. Continual testing during development plays a strategic role for successful software projects, allowing corrections to be made early on, before they become expensive and avoiding major disasters. The methodology described here calls for three kinds of testing during development: usability testing, to make sure that the product accurately reflects real user needs; class-level testing, on individual software units; and system testing, on the interaction between software classes as they work together.

- 
4. *Parallel development.* To the extent that available manpower allows, the below methodology is designed to support clean parallelization of development. For instance, the use cases and the class diagrams can be developed to some degree in parallel; and after identifying common functionality, different use cases can be implemented in parallel.

2.2 Implementation Procedures

JAM has the following key actions. Section 3 will show how these actions can be mapped onto the standard RUP methodology and the Unified Modeling Language (UML).⁵⁶

1. *Identify the end-users.* Understand the major categories of people that will use the product, and generally describe their needs.
2. *Explicitly define user needs.* This will be done with use cases. Each use case describes in simple language a major task that one or more kinds of users will need to perform with the product, and how they will interact with the product to complete the task. In addition to being a tool for developers, use cases provide a means to check with end-users that the correct functionality has been identified.
3. *Define and implement the user interface.* With an understanding from the use cases of the user tasks that the product will need to support, design and implement a user interface that will support these user tasks. The interface implementation provides a communication medium between developers and end-users, to make sure during development that the product is being built out correctly to meet user needs.
4. *Define the software functionality to support the use cases.* Given the use cases and the interface implementation, identify and define the set of software classes that are needed to support user tasks.
5. *Define the detailed relationship between the use cases and the software classes.* The use cases define the user view of the product; the classes define the software view of the product. Sequence diagrams are used to define the detailed relationship between the two views. Sequence diagrams define the dynamics of how the software classes interact to support the use cases.
6. *Iteratively build and test the product.* Identify the most important use case or set of use cases. Implement the software classes necessary to support the use case(s). Test the results, in terms of the functionality of the individual classes, the functionality of the interaction between the classes, and the usability of the results. Then build out the next most important use case(s), etc.

⁵ http://en.wikipedia.org/wiki/Unified_Modeling_Language

⁶ <http://es.wikipedia.org/wiki/UML>



The JackBe Application Methodology

3.1 Overview

This section describes a particular way to integrate UCD into RUP, consistent with the principles discussed above.

RUP defines four overall development phases: inception, elaboration, construction and transition. Each phase has particular deliverables.⁷

- The Inception Phase establishes the business case and high-level product requirements.
- The Elaboration Phase analyzes the problem domain and begins to define the software architecture.
- The Construction Phase builds out the product.
- The Transition Phase transfers the product from the development organization to the end-users.

Within each phase, multiple iterations may occur. Each iteration consists of identifying a critical problem needing to be solved, working on that problem, and test/evaluation of the results.

Activities can be organized in terms of disciplines (also called workflows).⁸ The outcome of activities, including documents, models and software, are called artifacts.

RUP requires the disciplines of business modeling, requirements, analysis and design, implementation, test, deployment, project management, environment and configuration and change management.⁹ Depending on the size of the project, some of these disciplines may be covered by the same people.

- Business Modeling. Develop the Business Case Document. At least informally, identify the business workflows related to the use of the product (for larger projects, these should be recorded as high-level use cases).
- Requirements. Identify the system use cases (also called requirement use cases). These describe user interactions with the product and collectively define the functional requirements. The system use cases must be sufficient to support the business model.
- Analysis and Design. Work out the software architecture and technology choices needed to support the system use cases.
- Implementation. Determine how the software classes will integrate with each other and with the user interface, and implement the classes.

⁷ See [Appendix A](#) for a table of deliverables.

⁸ <http://www.agilemodeling.com/essays/agileModelingRUP.htm>

⁹ <http://www-128.ibm.com/developerworks/rational/library/apr05/crain/index.html>

- Test. Develop Test Plan (including both software tests and usability tests for the interface), develop regression suites,¹⁰ conduct tests, analyze and communicate results, ensure the completeness of testing.
- Deployment. Deploy product, including the software itself, documentation, training and support.
- Project Management. Develop business case, make go/no go decisions, plan and manage iterations.
- Environment. Control the development process, determine project-specific guidelines.
- Configuration & Change Management. Establish change management policies, plan, and process, create a deployment unit, perform audits, review and manage change requests.

The following subsections describe how the RUP phases are executed within JAM. Because JackBe application development projects are expected to be relatively small and fast, JAM is a light-weight specialization of RUP.

3.2 Inception Phase

The inception phase establishes the business case and the high-level product requirements.

JAM requires that the following artifacts be in place before exiting the Inception Phase.

1. Business Case Document. Depending on the business and technical complexity of the proposed application, this may be a brief summary or a detailed analysis. Regardless, it should provide the following information:
 - Market overview. Description of the business problem the application will address.
 - Estimated project parameters. At least a rough estimate of the time, personnel, resources and overall cost to develop the application.
 - Estimated benefits. Benefits of developing the application.
2. Key Functional Requirements. This includes:
 - Actor profiles. Description of the kinds of users to be supported by the application.
 - Key use cases.¹¹ Each use case will describe how a user will interact with the application to carry out a particular task. These define the functional requirements of the application. Depending on the complexity of the application, these may be simply written task scenarios or may also include UML use-case diagrams describing relationships between the use cases.

¹⁰ http://en.wikipedia.org/wiki/Regression_testing

¹¹ http://en.wikipedia.org/wiki/Use_case#UML_Use_Case_diagram

3. Key Non-Functional Requirements.¹² Non-functional requirements describe further constraints on a system beyond support for particular user tasks. Non-functional requirements may cover such things as robustness, efficiency, security, extensibility, maintainability, portability, etc.
4. High-Level Interface Design. This should be sufficient to show the key properties of the interface envisioned. For a small project, the high-level interface design may be simply a block diagram description of key interface features. For more complex projects, it may include storyboarding,¹³ leading to navigational charts and/or state diagrams,¹⁴ and definitions of User Interface (UI) elements such as widget types, etc.
5. High-Level Software Architecture. This should be a proof-of-concept¹⁵ providing reasonable confidence that it is understood how to build a system consistent with the key requirements. For a small project, this may be a brief description of the key software components expected to be necessary in the application implementation and how they will interact. For more complex projects, it may involve UML class diagrams¹⁶ or other tools.

Before exiting the Inception Phase, the proposed project must pass business and technical review of the above artifacts.

Table 1: Inception Phase Activities

Discipline	Inception Phase Activities
Requirements	<ul style="list-style-type: none"> • Develop actor profiles • Develop key system use cases
Analysis & Design	<ul style="list-style-type: none"> • Develop high-level interface design • Identify key non-functional requirements • Develop high-level software architecture
Implementation	<ul style="list-style-type: none"> • Identify potential implementation leader
Test	<ul style="list-style-type: none"> • Identify potential test leader

¹² http://en.wikipedia.org/wiki/Non-Functional_Requirements

¹³ http://en.wikipedia.org/wiki/Web_Design

¹⁴ <http://www.developer.com/design/article.php/2238131>

¹⁵ http://en.wikipedia.org/wiki/Proof_of_concept

¹⁶ http://en.wikipedia.org/wiki/Unified_Modeling_Language

Deployment	<ul style="list-style-type: none"> • Develop understanding of who the product will be deployed to, and what support they will need
Project Management	<ul style="list-style-type: none"> • Participate in business modeling activities • Identify key personnel and other resources that will be needed if the project goes forward • Establish relationships with the stakeholders
Environment	<ul style="list-style-type: none"> • Identify potential environment leader • Develop consensus on the development process
Configuration & Change Management	<ul style="list-style-type: none"> • Identify potential CCM leader • Develop consensus on configuration and change management policies


3.2 Elaboration Phase

The Elaboration Phase analyzes the problem domain from the perspective of addressing technical risks and refining the software architecture.

The Elaboration Phase is conducted in a series of one or more iterations. An iteration is a development cycle that includes the stages of analyzing key problems that need to be solved, developing solutions the problems, and testing the results.

Consistent with its emphasis on UCD, the JAM goes beyond most RUP specifications in requiring that

1. The user interface be implemented by the end of the Elaboration Phase. During the Construction Phase, the application will be built out by successively implementing functionality to support the evolving user interface.
2. The user manual and training materials, based on the use cases, will be started during the Elaboration Phase. As with the interface implementation, this provides a tool to get early end-user feedback on the application design, and to support early marketing efforts. In the JAM, the user manual leads the software implementation, not the other way around. It is the responsibility of the software implementation to be consistent with the user manual.
3. Some level of usability testing should be a core part of the Testing Plan.



JAM requires that the following artifacts be in place before exiting the Elaboration Phase.

1. *Project management configuration*. A tool for managing the software project (possibly Artifact Software's Lighthouse).¹⁷ Aside from project management needs, this tool should be used to track project costs.
2. *Overall development plan*. Description of personnel requirements, estimated schedule, etc. required to build the application.
3. *Use cases*. The use cases should be at least 80% complete by the end of the Elaboration Phase.
4. *User manual*. Based on the use-cases, initial draft of the user manual and training materials for the application.
5. *Software architecture*. A fairly detailed description of the software components and their inter-relationships (possibly using class diagrams¹⁸). An initial demonstration (possibly using sequence diagrams¹⁹) that the software architecture is in principle sufficient to support the use cases.
6. *Integration Plan*. A description of how the application will work with other related software necessary to satisfy the business need.
7. *Test Plan*. Description of procedures and tools for testing the application during development. This should include both software testing (that the software is defect-free) and usability testing (that the software accurately reflects user needs). Usability testing can vary in scope from informal review by experts in the application's business domain to analysis of user task performance with the developing implementation of the application. The Test Plan should include provisions for automated regression testing.²⁰
8. *Deployment Plan*. Description of procedures and tools for transitioning the application to the end-user.
9. *Change Management Plan*. Description of procedures and tools for reviewing and managing changes, and for enforcing compliance with requirements.
10. *Interface implementation*. An initial version of a user interface sufficient to support the use cases.
11. *Business glossary*. Definition of common business terms specific to the application's target market.
12. *Technical glossary*. Definition of relevant technical terms.

During the Elaboration Phase, the artifacts produced during the Inception Phase should continue to be updated as appropriate.

The following table provides Elaboration Phase activities for each discipline.

17. <http://www.artifactsoftware.com/products/tools.html>

18. http://en.wikipedia.org/wiki/Class_diagram

19. http://en.wikipedia.org/wiki/Sequence_diagram

20. http://en.wikipedia.org/wiki/Regression_test

Table 2: Elaboration Phase Activities

Discipline	Elaboration Phase Activities
Business Modeling	<ul style="list-style-type: none"> Review the user interface, use cases and user manual as they develop for business relevance. Develop Business Glossary
Requirements	<ul style="list-style-type: none"> Develop system use cases Develop User Manual²¹ Develop user interface²²
Analysis & Design	<ul style="list-style-type: none"> Develop software architecture Select implementation technologies Develop Technical Glossary
Implementation	<ul style="list-style-type: none"> Develop Integration Plan.
Test	<ul style="list-style-type: none"> Develop Test Plan
Deployment	<ul style="list-style-type: none"> Develop Deployment Plan
Project Management	<ul style="list-style-type: none"> Create Development Plan Develop Project Management Configuration Develop Technical Glossary
Management	<ul style="list-style-type: none"> Develop Project Management Configuration Develop Technical Glossary
Environment	<ul style="list-style-type: none"> Develop project-specific guidelines, consistent with this document Communicate development process to team members
Configuration & Change Management	<ul style="list-style-type: none"> Develop Change Management Plan Setup Change Management environment

21. Traditionally, developing the User Manual would be seen as a Deployment activity. From a UCD perspective, however, the User Manual is part of clarifying and communicating the system requirements. By the "User Manual" is meant all materials needed for a new user to learn to use the finished product: this may include specialized training documents and interactive help.

22. The user interface could be put under the Implementation discipline. However, it is put under Requirements here to emphasize that the user interface is a representation of the functional requirements.

3.4 Construction Phase

The Construction Phase builds out the application in a series of iterations.

Consistent with its emphasis on UCD, the JAM goes beyond most RUP specifications in requiring that

1. The user interface implementation is the highest priority. The interface should be maintained at all times as a complete and accurate representation of the proposed user functionality to be supported.
2. As new functionality is implemented, it should be immediately incorporated into the interface implementation.
3. Each iteration should include a usability review. This may be as simple as an informal review by domain expert(s), or may include performance tests with potential end-users.

JAM requires that the following artifacts be in place before exiting the Construction Phase.

1. Completed Application. Tested and consistent with all requirements.
2. Technical Documentation. Documentation to support software maintenance (in addition to the User Manual)

Before each iteration, the key problems needing to be addressed are identified.

1. The highest priority will be on maintaining a complete interface implementation that accurately reflects functional requirements, as determined by the use cases.
2. If the interface implementation is up-to-date, use cases are selected for implementation in order of architectural importance. The use cases that will have the most impact on the design of the application are selected first for implementation. An iteration should touch all or most of the major subsystems of the application. A logically-related set of use cases may be implemented together in one iteration, or an iteration could be devoted to a single use case.
3. Iteration workload is picked for a relatively short expected completion time. The goal may be from one week up to about four weeks. However, completion times a measured variable, not a scheduled milestone.
4. A short Iteration Plan is produced, describing goals, expected resource requirements, and the start date.²³

During each iteration, the goals of the Iteration Plan are addressed. Each iteration will

1. Produce working code addressing the iteration goals.
2. Produce applicable technical documentation for the code produced.²⁴
3. Test the new code, in accordance with the Test Plan, including the creation of applicable regression testing materials.

²³ For a possible template, see <http://www.tcrug.org/Documents/Iteration%20Plan%20Template.doc>

²⁴ Note that since in JAM the user manual precedes code development, writing the user manual is not an issue during or after iterations. Of course, development may find problems during implementation that require the user manual (and possibly even the use cases themselves) to be updated.

4. Update the interface implementation to execute the new functionality produced.

In the event that the goals of the iteration are found to be too ambitious, the iteration can be refocused or cancelled. However, iterations should be planned to be sufficiently modest in scope that this is not a common occurrence.

The completion date of an iteration is a measured variable, not a scheduled milestone. The iteration completes as soon as it meets its goals, and not before. Actual completion time data is then used to update the estimated project completion date, which is derived from the measured average iteration time multiplied by the estimated number of iterations remaining.²⁵

During the Construction Phase, the artifacts produced during prior phases should continue to be updated as appropriate.

The following table provides Construction Phase activities for each discipline.

Table 3: Construction Phase Activities

Discipline	Construction Phase Activities
Business Modeling	<ul style="list-style-type: none">Review the user interface, use cases and user manual as they develop for business relevance.
Requirements	<ul style="list-style-type: none">Refine system use casesRefine User ManualRefine user interface
Analysis & Design	<ul style="list-style-type: none">Refine software architectureEnsure compliance between the software architecture and the requirements
Implementation	<ul style="list-style-type: none">Build out software classes, in accordance with Iteration PlansDevelop technical documentation, as the software is writtenOversee integration between software classesOversee integration between the user interface and the back-endProduce the finished application
Test	<ul style="list-style-type: none">Perform software and usability testsDevelop regression testing capabilities
Deployment	<ul style="list-style-type: none">Build training and support capabilities

²⁵ <http://www.objectmentor.com/resources/articles/RUPvsXP.pdf>

Discipline	Construction Phase Activities
Project Management	<ul style="list-style-type: none"> • Create and manage Iteration Plans
Environment	<ul style="list-style-type: none"> • Oversee development process, under direction of Project Management
Configuration & Change Management	<ul style="list-style-type: none"> • Review and manage change requests • Audit compliance with requirements

3.5 Transition Phase

The Transition Phase transfers the application from the development organization to the end-users. For JackBe applications the Transition Phase may not end, since ongoing maintenance may be required.

1. Transition Phase artifacts/activities include
 - Creation of the Deployment Unit, including
 - Production-quality software
 - User documentation
 - Technical documentation
 - Customized training
 - Ongoing support
 - Ongoing maintenance

2. Acceptance testing with the client and with potential end-users

During the Construction Phase, the artifacts produced during prior phases should continue to be updated as appropriate.

The following table provides Transition Phase activities for each discipline.

Table 4: Transition Phase Activities

Discipline	Transition Phase Activities
Business Modeling	<ul style="list-style-type: none"> • Participate in Acceptance Testing
Requirements	<ul style="list-style-type: none"> • Track testing results for possible defects, or for new or unmet requirements • As appropriate, request software maintenance • If major changes are required, request a new development project



Discipline	Transition Phase Activities
Analysis & Design	<ul style="list-style-type: none">• Track design-related user feedback
Implementation	<ul style="list-style-type: none">• Perform software maintenance as necessary
Test	<ul style="list-style-type: none">• Conduct Acceptance Testing• Track user requests for new or modified features• Track defect reports
Deployment	<ul style="list-style-type: none">• Oversee application deployment, ensuring adequate training and support
Project Management	<ul style="list-style-type: none">• Oversee Deployment• Review and approve/disapprove maintenance requests
Environment	<ul style="list-style-type: none">• No Transition Phase activities
Configuration & Change Management	<ul style="list-style-type: none">• Track changes occurring through software maintenance

Appendix: Table of Artifacts

Table 5: Project Artifacts

Phase	Description	Deliverables
Inception	<ul style="list-style-type: none">• Understand business and technical requirements• Design high-level architecture	<ul style="list-style-type: none">• Business Case Document• Key functional requirements, including actor profiles and use cases• Key non-functional requirements• High-level interface design• High-level software architecture
Elaboration	<ul style="list-style-type: none">• Refine requirements• Develop detailed and validated design• Refine implementation plan	<ul style="list-style-type: none">• Project management configuration• Development Plan• Use Cases (80%+ complete)• Draft User Manual• Software architecture• Integration Plan• Deployment Plan• Change Management Plan• Test Plan• Interface implementation• Business Glossary• Technical Glossary



Phase	Description	Deliverables
Construction	<ul style="list-style-type: none">• Iterative and incremental implementation, integration, and testing of applications	<ul style="list-style-type: none">• Iteration Plans• Regression-testing capabilities• Technical documentation• Working software
Transition	<ul style="list-style-type: none">• Conduct acceptance testing• Training• Prepare and launch production• Operation & support	<ul style="list-style-type: none">• Deployment Unit