

Iterative Implementation Strategy

Stephen McHenry

Copyright 1997 - Advanced Software Technologies, Ltd. (<http://www.softi.com>)

All Rights Reserved

Permission to reproduce and distribute is hereby granted for non-commercial purposes (that is, it cannot be sold or included in a collection to be sold), provided that it is copied and distributed in its entirety.

An explanation of implementing systems using a strategy of iterative implementation and continuous integration.

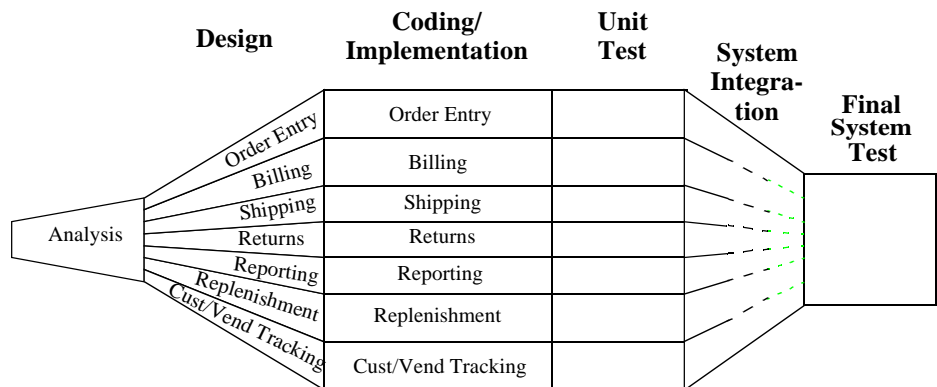
This document describes how to implement software systems using an iterative implementation strategy combined with continuous integration.

Traditional Development

In the past (and even still today), many software development efforts have utilized an implementation approach which develops major subsystems in parallel, and integrates them at the end of the development process just prior to final system testing. Systems developed with big bang integration often utilize a waterfall approach to analysis and design, as well. Typically, the overall flow for these systems is an analysis phase, during which the overall system is broken into subsystems based upon functional area. Examples of these functional areas might include Order Entry, Billing, Shipping, Returns, Replenishment, Customer/Vendor tracking, and Reporting.

Once these subsystems have been identified, work for the remainder of the project splits and is organized based upon subsystems. Except at well defined interface points, these subsystems are designed and developed in isolation until it comes to the system integration phase of the project. This approach is depicted graphically below.

Iterative Implementation



Problems with “Big Bang”

This strategy, sometimes known as “big bang” integration, has traditionally experienced some of the following problems:

- subsystems are developed using fundamentally different assumptions, requiring major rework on the part of one or more teams. Since this occurs at the end of the implementation cycle, just prior to delivery, major rework dooms the project schedule. Additionally, since no team wants to be the one that must rework their area, the actual rework (which often takes months) is often preceded by weeks or months of denial, followed by weeks or months of argument about who’s subsystem is “at fault” for not “fitting” (because it’s their subsystem that must be rewritten to conform to the “correct” one).
- Often, substantial “test harness” code must be developed to simulate the functionality that is provided by other subsystems.
- Little, if any, of the final product is demonstrable until just before it is finished.
- Most testing cannot be done until the integration phase is complete.
- Implementors are sometimes left with the “Are we ever going to be done?” feeling.

Iterative Implementation

There is another implementation approach that eliminates most or all of these problems. It is known as the Iterative Implementation/Continuous Integration Approach. This is also known as Design/Implementation Increments.

With this approach, the overall functionality of the system is broken down into feature sets. These feature sets, often based upon use cases from the analysis stage, contain a group of individual features that are related, typically by functional area. In most cases, the entire set of features required to support a functional area will be too large to be considered a single feature set, and must be further divided to reduce the size (more on scoping later).

To make this a more real example, let's consider this in the context of a company whose business is to sell both products and services to its customers. Products are tangible things that must be shipped to customers. Services include installation (one time only) and monthly services. (There are many companies that have this as a business profile. For this example, let's consider one that operates in the Cable TV market. They sell TVs, descrambler boxes, etc. They install them plus cable service and they charge monthly for the service.) In addition, we need to keep track of connections that have been made by the installers so that additional connection points can be provided, if required.

System functions:

- New customer orders shippable product(s) only
- New customer orders installable product(s) only
- New customer orders service(s) only
- New customer orders combination of shippable product(s), installable product(s) and service(s)
- Generate monthly billing
- Existing customer orders additional product(s)
- Existing customer adds service(s) only
- Existing customer cancels service(s) only
- Customer returns product(s)
- Customer changes service address
- Customer changes billing address
- Customer changes order
- Customer cancels order
- Receive and apply customer payments
- Correct Customer bills/billing and regenerate bills

This represents the list of “business” functions that we need to accomplish with this system. However, because things often don't occur as planned, we also need to account for anomalous conditions like products being out of stock when they are ordered, general product shortages (resulting in allocations), inadequate capacity to add new lines (for service to new customers), etc. As a result, our system must provide additional functionality to accommodate these situations. These include:

- Creating back orders
- Shipping back ordered products
- Allocating products to customers
- Adding new lines for service
- Reschedule installation (customer wasn't home, or technician didn't have correct parts, etc.)
- Terminate a “no longer offered” service

Iterative Implementation

The features/feature sets that might be developed on the first pass are:

Function	Required Action	Functional Area
New customer orders shippable product(s) only	Add Customer	Customer Maintenance
	Enter Order Information	Order Entry
	Create Packing Slip	Shipping
	Prepare/Ship Shipment	Shipping
	Send bill to customer	Billing
New customer orders installable product(s) only	Add Customer	Customer Maintenance
	Enter Order Information	Order Entry
	Schedule Installation	Order Entry
	Install Product	Installation
New customer orders service(s) only	Add Customer	Customer Maintenance
	Enter Order Information	Order Entry
	Allocate Resources	Order Entry
	Schedule Installation	Order Entry
	Install Service	Installation
New customer orders combination of shippable product(s), installable product(s) and service(s)	Add Customer	Customer Maintenance
	Enter Order Information	Order Entry
	Allocate Resources	Order Entry
	Schedule Installation	Order Entry
	Create Packing Slip	Shipping
	Prepare/Ship Shipment	Shipping
	Install Service/Product(s)	Installation
	Send bill to customer	Billing
Generate monthly billing	Generate Bills	Billing
Existing customer orders additional product(s)	Enter Order Information	Order Entry
	Create Packing Slip	Shipping
	Prepare/Ship Shipment	Shipping
	Send bill to customer	Billing
Existing customer adds service(s) only	Enter Order Information	Order Entry
	Allocate Resources	Order Entry
	Schedule Installation	Order Entry
	Install Service	Installation
Existing customer cancels service(s) only	Enter Order Information	Order Entry
	Cancel Service	Installation
	Deallocate Resources	Installation
Customer returns product(s)	Issue RMA	Customer Service
	Receive Product(s)	Shipping
	Credit Customer Account	Billing

Iterative Implementation

Function	Required Action	Functional Area
Customer changes service address	Modify Customer Information	Customer Maintenance
Customer changes billing address	Modify Customer Information	Customer Maintenance
Customer changes order	Modify Order Information	Order Entry
Customer cancels order	Cancel Order	Order Entry
Receive and apply customer payments	Apply payment to customer account	Billing
Correct Customer bills/ billing and regenerate bills	Modify customer bill	Billing
	Generate Updated Bill	Billing
Customer orders products which are out of stock	Enter Order Information	Order Entry
	Create Back Order	Order Entry
Shipping back ordered products	Receive shipment	Shipping
	Allocate stock to back orders	Order Entry
	Create Packing Slip(s)	Shipping
	Prepare/Ship Shipment(s)	Shipping
Allocating products to customers (Customer buys product which is “on allo- cation”)	Enter Order Information	Order Entry
	Allocate product to customer	Order Entry
	Create Packing Slip	Shipping
	Prepare/Ship Shipment	Shipping
Adding new lines for ser- vice	Enter Order Information	Installation
	Schedule Installation	Installation
	Install Equipment	Installation
Reschedule installation (customer wasn’t home, or technician didn’t have cor- rect parts, etc.)	Contact customer & reschedule	Installation
	Install product or service	Installation
Terminate a “no longer offered” service	Determine Customers affected and send cancellation letter	Order Entry
	Enter Order Information for each customer	Order Entry
	Cancel Service	Installation
	Deallocate Resources	Installation

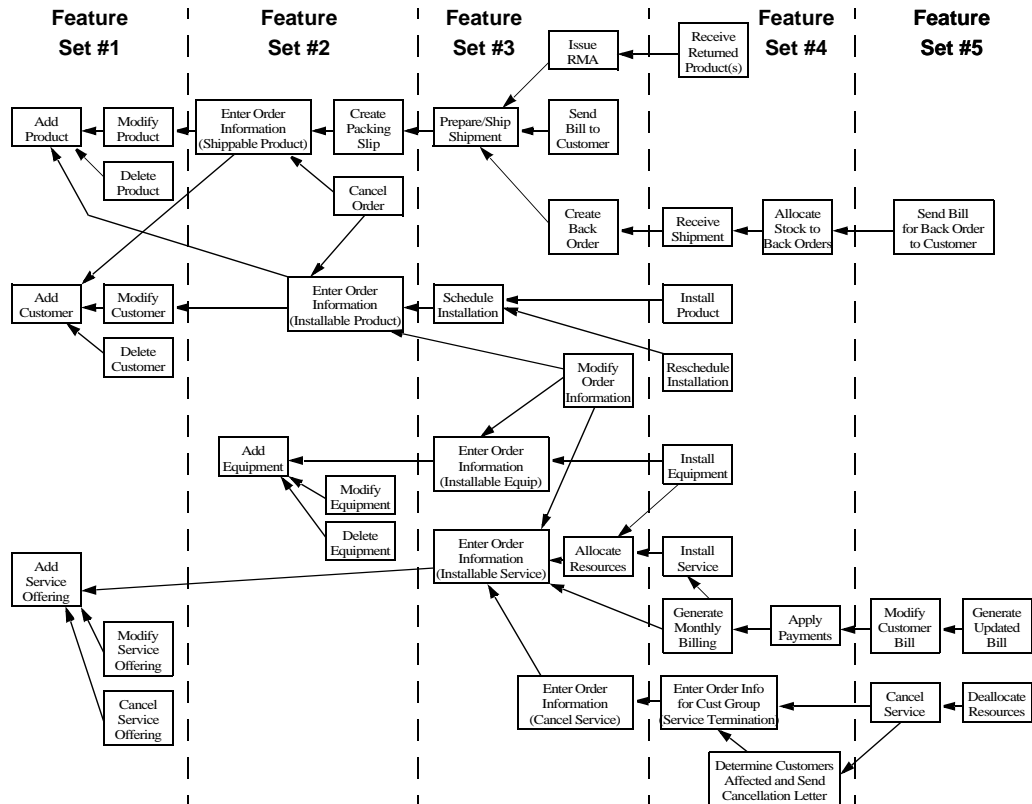
Iterative Implementation

Next, the responsibilities of each functional area are grouped to give a clear picture of what that functional area needs to do. An example of this is shown below.

Functional Area	Required Action
Billing	Apply payment to customer account
	Credit Customer Account
	Generate Bills
	Generate Updated Bill
	Modify customer bill
	Send bill to customer
Customer Maintenance	Add Customer
	Modify Customer Information
Customer Service	Issue RMA
Installation	Cancel Service
	Contact customer & reschedule
	Deallocate Resources
	Enter Order Information
	Install Equipment
	Install Product
	Install Service
	Schedule Installation
Order Entry	Allocate product to customer
	Allocate Resources
	Allocate stock to back orders
	Cancel Order
	Create Back Order
	Determine Customers affected and send cancellation letter
	Enter Order Information
	Enter Order Information for each customer
	Modify Order Information
	Schedule Installation
Shipping	Create Packing Slip
	Prepare/Ship Shipment
	Receive Product(s)
	Receive shipment

Next, the feature sets are examined for dependencies and a dependency graph is produced. To keep from writing “throw away” code, the features that have no dependencies

are implemented before those that are dependent upon them. So, the amount of code that is being produced that is not actually deliverable code is kept to a minimum. An example of this is shown on the following page.



Then, individual features are grouped together for simultaneous development. These are depicted by the vertical lines in the diagram.

One of the purposes of the feature set development approach is to integrate all of the components with all previously existing components while the initial development is being done.

Feature Set Criteria

Each iteration should have several characteristics. These include:

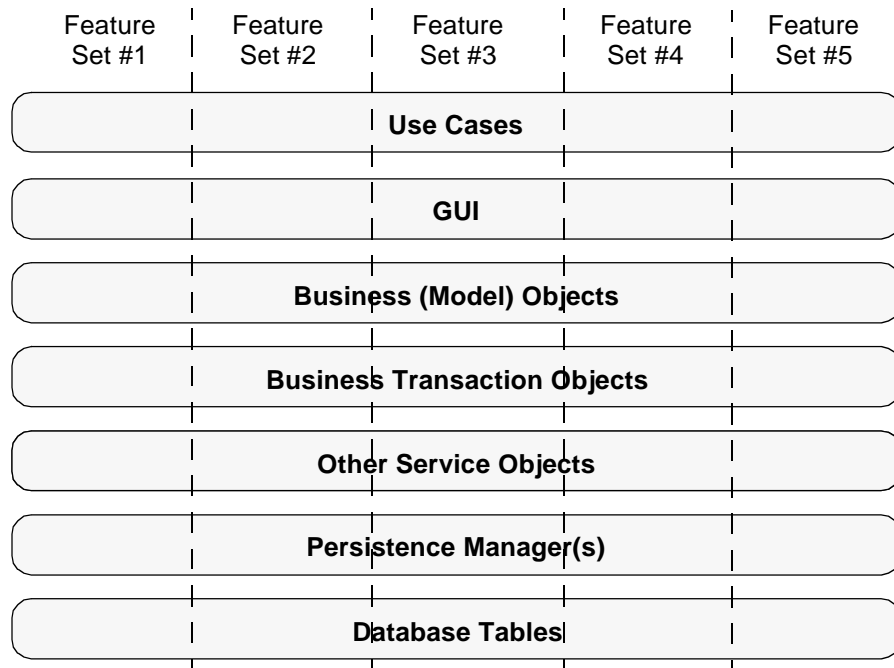
- It should take about 6-8 weeks to implement
- It should depend only upon code that has already been implemented. (i.e., it should not require “test harness” code to be written.)
-

Iterative Implementation

When developing an iteration, all of the different parts of the system need to be completed to call the iteration complete. This includes:

- Use Cases
- GUI Screens
- Business (Model) Objects
- Business Transaction (Policy) Objects
- Other Service Objects (Security, Logging, etc.)
- Persistence Managers
- RDBMS Tables, Indices, Triggers, Stored Procedures, etc.

This is shown graphically below.



When the iteration is complete, all of the portions will be operational and can (if desired) be turned over to testing and/or documentation groups.

It is important to realize that each iteration may not produce a system which is complete enough to use. Furthermore, it may be the case that facilities other than the system need to be used to verify that the iteration performed properly (i.e., if an iteration includes the “Add Customer” functionality, but not “Retrieve Customer”, it may be necessary to use SQL to interrogate the database to verify that the customer was successfully added. This is fine. It does not mean that the iteration is incomplete or unfinished.)

Iterative Implementation

Duration of Each Iteration	<p>Each iteration should require approximately 6-8 weeks for implementation. There is a certain overhead for beginning and finalizing each iteration, and shorter iterations incur this overhead more often, making it a larger percentage of the project. Also, it is more difficult to actually implement a meaningful portion of a large project in less time.</p> <p>Iterations should not be allowed to grow much beyond the 6-8 week figure either. If an iteration seems to require 10-12 weeks, it is probably better to split it into two iterations.</p>
Purpose of Iterations	<p>There are a number of things that this implementation strategy is specifically designed to achieve. These include:</p> <ul style="list-style-type: none">• Forces a continuous integration of all components, so issues surrounding integration are resolved painlessly (or at least with minimal pain)• Forces design issues to be resolved rather than thrashed forever without resolution.• Allows mistakes made (and knowledge gained) in early iterations to be used by later iterations.• Refocuses developers from the abstract nature of analysis and (some) design to the concrete nature of implementation.
Differences From Traditional Development	<p>There are a couple of principal differences from traditional development. First, there is no “design complete” moment when all low level design documentation has been produced and can be reviewed. Instead, this low level design is produced on an ongoing basis during the entire implementation cycle. And, while reviews are still appropriate, the structure and breadth of the review process must be modified to realistically mesh with the iterative approach.</p> <p>There is also no need for a “system integration” phase of the project. When the last line of code has been written and integrated (during the last iteration), the system is fully integrated.</p>
Similarities with Traditional Development	<p>There are also a number of things that have not changed, just because the system is being developed in an iterative manner.</p> <p>It is still necessary to produce a System Requirements document detailing what the system needs to do. This is probably the most important document in the system lifecycle. It defines what must be built.</p> <p>Even when using an iterative development approach, it is still necessary to do analysis & design. Analysis - understanding the problem to be solved - is essential before you can begin to think about building a solution. Design - mapping that understanding to a specific solution using a specific set of tools within a given set of constraints - must also be done to ensure that the proposed system does not encounter undue rework due to lack of understanding or inadequate planning.</p> <p>And, all systems need thorough testing, regardless of how they were developed. Iterative implementation does not alleviate the need for comprehensive testing.</p>
Advantages	<p>Advantages of the Iterative Implementation approach include:</p>

Conclusion

- Eliminates the large risk of “big bang” integration - i.e., that major subsystems will be developed using fundamentally different assumptions, requiring rewrite of one or more.
- Allows functional testing of the (earlier stages of the) project in parallel with later stages of development
- Allows performance testing as phases are completed, enabling identification and correction of performance bottlenecks
- Eliminates the need for most (if not all) “test harness” code.
- Allows documentation to begin prior to completion of the project
- Allows training to begin prior to completion of the project (if necessary).
- Allows phased deployment - the system can actually be rolled out to users for early acceptance testing and other deployment issues.
- Allows “lessons learned” in early steps to be used in the design of later steps
- Allows detailed scheduling of the implementation phase, allowing identification of critical resource shortages
- Allows good visibility into the progress of the implementation phase
- Increases confidence of those external to the project (management and users) that delivery will happen
- Manages that difficult transition from design into implementation
- Helps get teams “going” at this critical stage
- Allows “post V1” enhancement in an incremental way.
- Helps build morale as implementation progresses.

Conclusion

The iterative implementation development strategy smooths the transition from analysis and design into implementation. It gets teams going and helps them produce tangible, repeatable results in the quickest, most cost effective manner with a minimum of rework. As such, it is probably the single most effective strategy in management’s arsenal for producing viable systems.