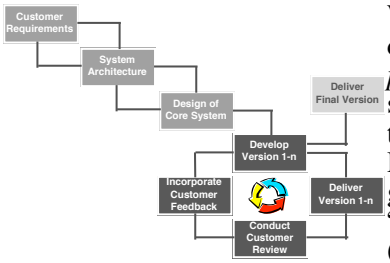
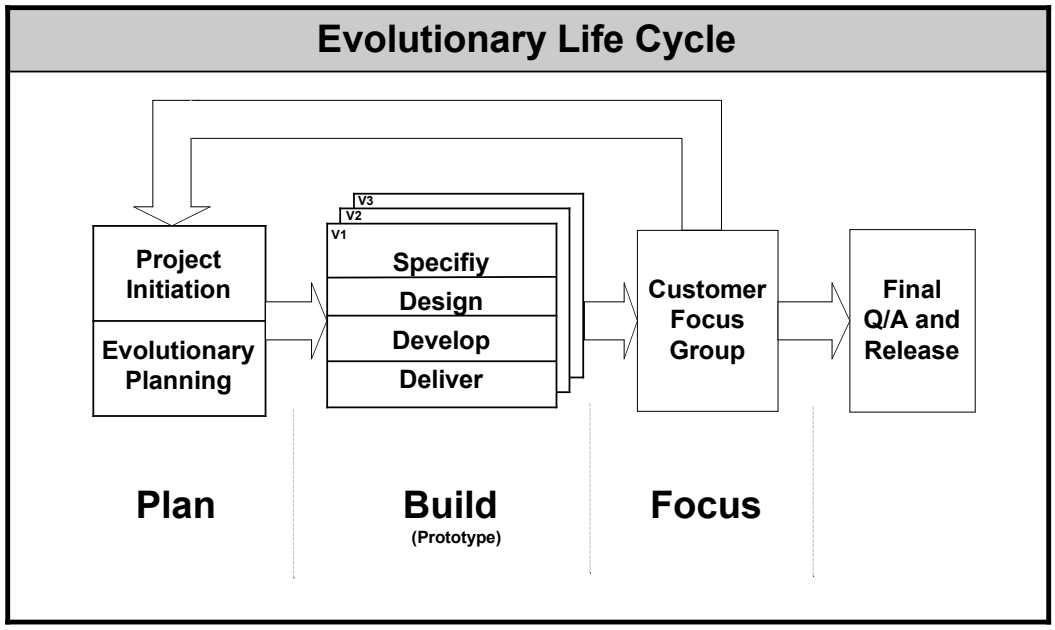


Whereas the waterfall life cycle's focus is on managing interim documents, the evolutionary delivery life cycle focuses on interim *products*. The difference is significant, although many aspects of that significance are subtle with awareness and understanding growing over time. Both development and management of this Evolutionary Development process are very different, and it is easy in the beginning to get a muddled understanding by over-concentration on traditional "task" lists. This guide concentrates first on defining product versions (this section) and then on defining what tasks need to be accomplished for each product version (next section).



The Evolutionary Life Cycle

Evolutionary Version Plan

Objective

The objective of the version plan is to establish how the project, or projects if necessary, will be organized. There are two basic questions to be answered in the version plan. First, is the application large enough to be divided into multiple segments, and if so what are the boundaries of each? Second, how many versions will be needed, and what will be the emphasis of each?

TechNote -- Evolutionary Process Concepts

The term Evolutionary Development was defined, at least in the manner it is used in this framework, by Tom Gilb in his book *Principles of Software Engineering Management*. The basic tenets of Evolutionary development presented here are derived from Gilb's principles.

- Customer orientation
- Total product approach
  - Include the entire range of application/product deliverables, not just code. Things like -- documentation, training, marketing literature.
- Early, frequent iteration of product versions
  - Versions should be weeks, not months.
  - Select the potential steps with the highest value to cost ratio for earliest implementation.
- Risk-driven iteration cycles
- Make mistakes, don't do it right the first time
  - Plan to do it over several times. "Good enough" is OK.
- Open-ended basic systems architecture
  - One of the high risk factors of evolutionary development is getting locked into an inappropriate architecture early in the effort.
  - Attention to architectural issues must be an ongoing task.
- Results, not process orientation
  - In a traditional waterfall model software development cycle, the process itself often seems more important than the result.
- High performance development tools are necessary

Deliverables

- Application Segmentation Plan
- Evolutionary Version Plan
  - Product/Feature Description for each Version
  - Project Task List and Schedule by Version

Process Description

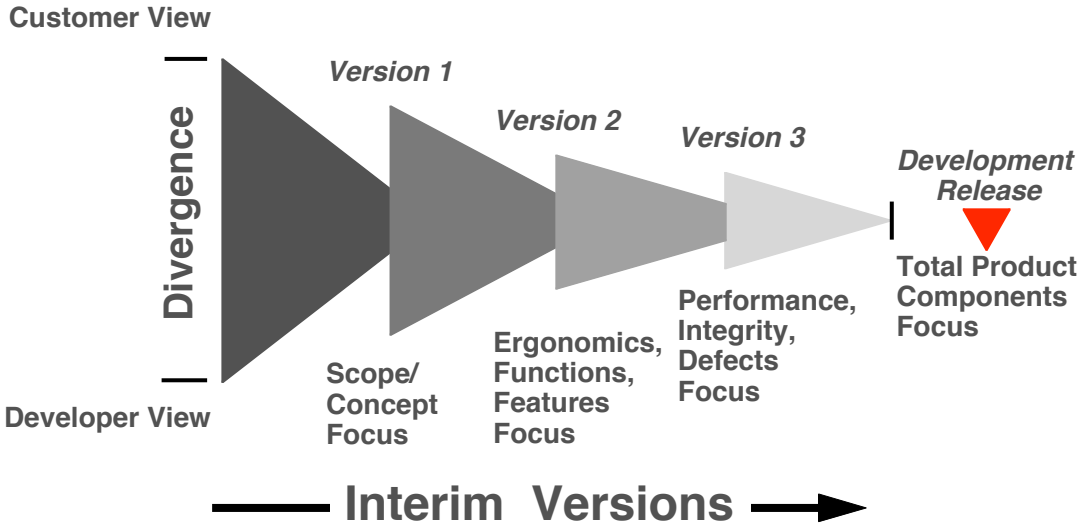
Planning a Evolutionary Development project is very different than planning a traditional life cycle one. The process is essentially two dimensional rather than one dimensional. The dimensions are product versions and then tasks, whereas more traditional planning involves mostly task planning. In a Evolutionary Development project, the team must first define the final product components in detail, then allocate those components to a version of the development cycle, and finally plan the tasks necessary to implement each version. This section describes techniques for approaching this "two dimensional" planning.

Before describing the process of evolutionary version planning, a definition of the terms version and build are in order.

- Version -- a product version is a working, demonstrable portion of the application. Furthermore, it should be the subject of a formal Customer Focus Group review. Any defined version which does not have a customer reviewable component should be viewed very critically. In a Evolutionary Development project, versions may be produced on a 1 week (small project) to a 4 week (6 month project) cycle.

- Build -- a product build is a working portion of the application, but is for the internal use of the Project Team and not a major customer review. Customer core team members may participate in reviewing builds, but a formal Customer Focus Group review is not done. Builds may run on a daily cycle close to version release time.

Version planning should be done in conjunction with the Initiation and Planning phases. The version plans will dictate the work breakdown structure (tasks) in the project plan. If the application is larger than the recommended Evolutionary Development project size it will require segmentation. This segmentation process would be similar to what Information Engineering calls Business Area Analysis (BAA). If a BAA has been done for this area, segmentation may not take much additional time. If a BAA, or equivalent, has not been done, the team may need to take some additional time to analyze and define the application segments by doing high-level process and data modeling. Details of this segmentation process are beyond the scope of this discussion.



If a segmentation process is conducted at this point, it should be accelerated and time-boxed. The objective is not a complete definition of the data model (for example), but only enough detail to assist in segmenting the application and integrating the components once built. As the number of segments per application increase, each Project Team (one per segment) will need to include additional task time for overall architectural integration.

Based on the project's estimated size, the final delivery date is set. This could be anywhere from one to a maximum of nine months with an optimum period of under six months. The next step is to define how many versions will be required. A six month project might have from four to seven versions depending on the nature of the project. The first few versions should be no longer than 3-4 weeks apart to insure adequate customer feedback. Later versions perfecting performance, removing defects, and preparing for delivery may take slightly longer between versions. The next step is to define what each version's *product* will contain; and then to define what work needs to be accomplished in each version to produce that product.

The version plan should be revised at the end of each version. Since Evolutionary Development projects are geared to change, plans will probably require more revisions than traditional life cycle projects. The only thing that doesn't change, without team and Project Sponsor approval, is the final delivery date. One of the objectives of the first version is to confirm that the original project size estimate was reasonably accurate. If the original scope and size estimates prove to be significantly in error, the team needs to reevaluate the project and discuss with the Project Sponsor.

	V1.0	V2.0	V3.0	V4.0
Version Delivery Dates	1-Jun	1-Jul	1-Aug	1-Sep
Application				
Functions/Features	x	x		
Data	x	x		
User Interface		x		
Algorithms		x		
System Interfaces			x	
Error Handling			x	
Security & Control				x
Server				
Client/Servier Arch	x			
Server Architecture		x		
Client				
Client Architecture		x		
Communications Network			x	

Table - High Level Version Plan

The first version in which a component appears is the one in which most of the development for that feature will occur. Changes resulting from the Customer Focus Group or other sources will occur in the subsequent versions. Particularly complex features may span several versions, However, the Project Team needs to allow sufficient time for corrections after the first introduction of a feature.

Use the following guidelines to help determine what features to assign to what versions:

- Deliver usefulness to the customer in every version
- Focus on key business exchanges first
- Deliver feature breadth first, depth later
- Assign risky features to the early versions

A version plan consists of all the final components of the product, whether executable code or user documentation. The table shows how different dimensions of the product might be addressed if the application was developed in four versions. This table is also used to show where different product dimensions are described in this evolutionary process section. This allocation is for descriptive purposes only; the actual development of each of these dimensions will be different, based on individual project characteristics and its risk analysis. A further breakdown of a version plan is shown in the discussion of Version 2 where features and functionality are emphasized.

Once the version plan is established, the tasks necessary to accomplish each version can be identified. These would be tasks such as:

- Conduct the JAD session
- Develop the initial data tables
- Develop the initial navigational menus

Version Plan for ACME Ordering System

Project Deliverables	Concept Version 6/15/97	Features Version 7/15/97	Performance Version 8/07/97	Total Product Version 8/15/97	Release Version 9/15/97
Business Components					
System Menus	x	x	x	x	x
Order Entry Screens	x	x	x	x	x
Warehouse System Interface		x	x	x	x
Order Pricing Screens		x	x	x	x
Back Order Screens			x	x	x
Handling Pricing Errors			x	x	x
Secure Access			x	x	x
User Training Materials				x	x
User Manuals				x	x
Help Screens				x	x
Technology Components					
PowerBuilder Installed	x	x	x	x	x
Workstation	x	x	x	x	x
Server		x	x	x	x
Communication Lines				x	x
Warehouse Connections					x
Systems Documentation					
Business Context Model	x	x	x	x	x
Logical Data Model	x	x	x	x	x
Client/Server Strategy	x	x	x	x	x
System Architecture		x	x	x	x
Physical Database Schema		x	x	x	x
System Test Plan			x	x	x
Performance Test Plan			x	x	x
Data Conversion Plan				x	x
Installation Plan					x
User Acceptance Test Plan					x

Table - High Level Version Plan for ACME Ordering System

Another parallel concept for activities in this new Evolutionary Development approach is that analysis, for example, will occur during each version's development. It may be a greater effort in early versions,

but will occur in all versions as the task effort by version table shows graphically.

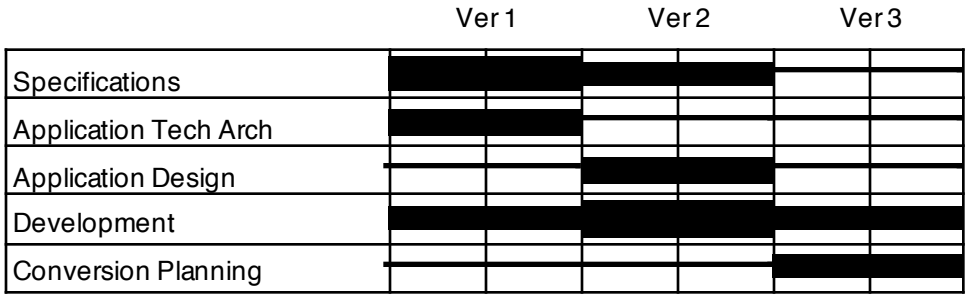


Figure - Relative Task Effort by Version

A software development project consists of much more than executable modules. Capers Jones has estimated that 20% to 30% (depending on product size) of development cost is paperwork related. While Evolutionary Development methods attempt to minimize formality of documentation to reduce delays and workload, some documentation is part of every project. So in addition to a "product" version plan, a Evolutionary Development project needs a "documentation" version plan also. Again, as project size increases, documentation needs increase also. The table illustrates a sample documentation version plan.

Activity	Document Example	V!	V2	V3	V4
Specify	Business Case	D	R		
	Data Model	B	D	R	R
	Functional Description	B	D	R	R
Design	Architecture Strategy	O	D	R	R
	Test Plan	O	B	D	R
	Conversion Plan	O	B	D	R
Develop	Help Screens		O	B	D
	Training Manual		O	B	D
	User Manual		O	B	D
Project Mgt	Project Data Sheet	D	R	R	

Table Legend

O	Outline
B	Breadth, something in each major topic
D	Depth, details in each topic, as needed
R	Revise and/or complete

Table - Documentation Versioning Example

Risk-Driven Version Planning

Risk analysis should be used in two ways: first, to assist in version and task planning, and secondly, to then modify the basic plan to manage the risks.

There is a long list of projects that have failed because risks were ignored until it was too late. In many cases, far too much time was spent on some activities and not enough on others. Given just a little emphasis on identifying the risks (they are usually well known, but no one wants to recognize them) and working to minimize the consequences, many of these projects could have been salvaged.

In planning a evolutionary project, it is usually a better strategy to place high-risk items in early versions. For example, mis-identification of scope is a common risk in projects. By concentrating on broad-based scope items like menu structures and data model validation in Version 1, this risk area is minimized. If a second area of risk was integration with some existing mainframe systems that all consisted of DB2 data bases, a good strategy might be to include at least one of those interfaces by Version 2.

In version planning, both product and documentation components should be considered.

- First, it will provide a basis for which product components of the application will be worked on during each version, both in terms of the breadth and depth of the component.
- Secondly, it will provide a basis for determining the completeness, formality, and granularity (depth) of documentation components of the application (for example, early version broad logical data modeling, later versions more in-depth physical data models).

Three categories of risk are relevant:

- First, risk to your project -- the primary objective is to get your project implemented on time.
- Secondly, risk to inter-related projects -- if your project is one segment of a multi-segmented application, there will be some overall risk reduction strategies that might slow your down somewhat, but will make the overall application more successful. An example would be more formal interface specifications.
- Thirdly, future risk -- are we compromising so much to get your project implemented, that a “mess” is left for future maintenance?

High-risk items should be identified, made very visible to the Project Team, and thought about every day.

Larger Projects

Larger projects, those consisting of multiple application segments and larger than about 1,500 function points, need special consideration. Even if larger projects are broken into “segments” with a team assigned to each, the time frames for development will be increased by the need for more “inter-project” communication and coordination. While each of these could be run as a Evolutionary Development subproject, the following points will help in managing the overall effort.

- More evolutionary planning work will be required up front. The larger the project, the more important segmentation will be to effective completion. Poorly segmented applications will either lead to significant time wasted in inter-project coordination or, in the worst case, extreme time delays due to lack of integration until late in the project.
- Architectural efforts, both in the initial planning and during project development, will need to be emphasized. For example, an overall application data model or object model needs to be developed and maintained by the “architecture” team. This architecture team might be a part-time coordination group with members from each subproject.
- Dependencies between subprojects should be identified and carefully managed. Dependencies are defined in the form of deliverables, either documents or product components. A

document that is an inter-project dependency will need to be more formal and detailed than an intra-project document.

- Communication lines between subprojects need to be carefully thought out during the Initiation phase.
- There should be an overall Project Data Sheet (including a project objective statement and a PDS for each subproject).
- Versions may be longer in larger projects. Versions should be used to synchronize subprojects.

Version 1: Scope and Concept

Objective

The focus of Version 1 is to verify the scope and concept of the project. In the Initiation phase the Project Team gathered existing information about the project, developed an initial project objective statement and started "ability to" statements about the features and functionality of the application. In Version 1, the team is striving to provide more depth of understanding to the project's objectives and business benefits. When this version is delivered, the team will be able to confirm with the customer organization and the Project Sponsor if the project is still viable and on-track with the original scope expectations.

The results of risk analysis in the Planning phase may in certain circumstances dictate that other factors are more important than scope and concept in Version 1. For example, in a new client-server application, it may be more important to get the network and basic hardware and software to work and leave the application scope until Version 2.

The customer team members have definite tasks to accomplish during the development process. They are heavily involved in developing the specifications and reviewing interim builds of the product. They are concentrating on developing business scenarios by which they can demonstrate the abilities, and test the system.

Deliverables

- Product Version 1
- Preliminary Documentation
  - Specifications
  - Data Model
  - Design
- Updated Evolutionary Version Plan
- Customer Focus Group Results

Process Description

The objective of this process is to convert the preliminary definition of the project into a rudimentary first version of the product and appropriate supporting documentation. A primary tool of this process will be Joint Application Development (JAD) sessions. Other requirements gathering techniques, such as building process and data models will be utilized both during, and apart from, JAD sessions. The development team then builds a version of the application. This version will probably *not* be very pretty. The application will then be submitted to a Customer Focus Group review. A technical review should also be done at this time.

TechNote -- Joint Application Development (JAD)

JAD sessions have proven to be one of the most effective requirements gathering techniques. They bring together the key users of the application in a well-structured, facilitated meeting where requirements can be uncovered, analyzed, refined, and documented very quickly. The meeting "structure" is provided by a selected set of modeling and requirements techniques organized to provide specific information needed for design and development. Because of the evolutionary prototyping capabilities, the requirements JAD session provides the basic framework for development, but not necessarily all the details.

Specify

In most cases, requirements definition will begin with a 1-3 day Joint Application Development (JAD) session. Prior to the JAD session, initial documentation would be gathered and analyzed. The composition of participants in the JAD session will depend on the type of project. During the JAD session, the team should concentrate on the items identified in the JAD preplanning session.

Modeling at this stage should be done at a general level. Details can be added, if necessary, during later versions. Examples of models appropriate at this point are:

- Context model -- all the internal and external stakeholders in the project.
- Process model -- one or two levels of data flow diagrams or equivalent. Concentrate on the “main-line” business process flow.
- Data model -- identify the key business entities and important relationships. Some attribution may be necessary to verify or define the entities, but extensive attribution, associate entities, etc. might be done in Version 2.
- State model -- usually required with applications with significant real-time components.
- Object model -- high-level object modeling techniques may replace some or all of the models above depending on the particular approach used.

The size, complexity, or organizational documentation requirements will be factors in how formally the models are documented. In some situations -- for example small stand-alone applications, flip-charts, wall boards and legal pads -- may be sufficient. For larger applications which impact other organizational systems, more formal documentation and/or CASE tools may be necessary.

One of the important products of the specify phase, regardless of approach used, is a list of “ability to” statements. These statements reflect the business processes that need to be implemented. In each phase these are further refined. On the Project Data Sheet, the high-level “ability to” statements are under the “Features” section. These “ability to” statements are relevant and important:

- To the customers as a statement of their requirements
- To developers as a version planning tool
- To the project manager as a progress measurement tool
- To the developers as a basis for testing and quality assurance

These statements can be depicted in an indented list, a process decomposition diagram, or a bracket diagram as shown below.

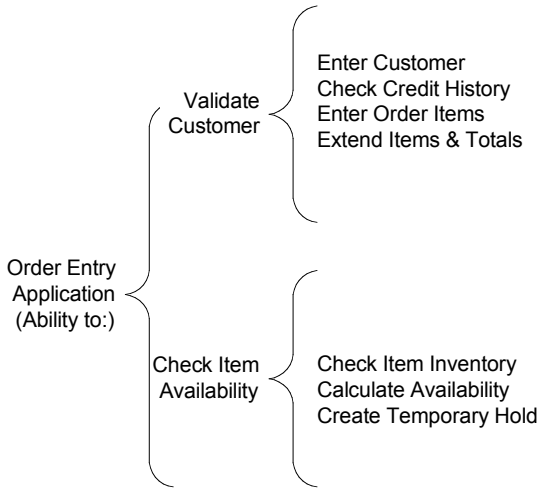


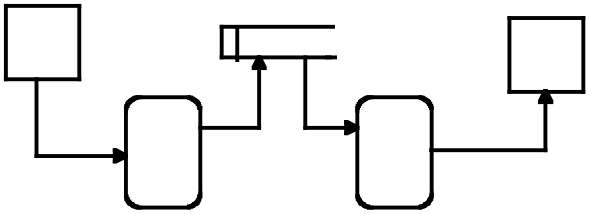
Figure - Ability To Bracket Diagram

In addition to the data/feature/object specifications, Version 1 needs to specify the basic client/server strategy. In keeping with recent industry trends, a mainframe development strategy will be considered as a sub-type of the server category.

In many projects, the basic architecture will be established prior to the start of the development effort. In either case, during Version 1 the basic specifications that will determine design alternatives need to be documented.

TechNote -- Data Flow Diagrams

Data Flow diagrams are the most widely used of the structured analysis modeling techniques. Whether the Yourdon, Gane and Sarson, or variations are used, they can be extremely valuable in getting the project started off right. Information Engineering adherents often leave data flow diagrams until later in the process and utilize process decomposition diagrams early in the specification process. We think DFDs are more useful for understanding the overall business application process flow.



Design

Design issues addressed during Version 1 are those concerned with overall architectural decisions and those having an impact on the products maintainability. Some of these architectural issues may have been decided during project planning, be constrained by an organization's technology architecture plan, or be constrained by existing products. The architectural or high-level design factors to consider during Version 1 are:

- Hardware (mainframe, PC, client/server configuration)
- Network (LAN, WAN)
- System software (operating system, network, DBMS)
- Development tools
- Integration strategy (API's, ODBC, OLE, SAA, SNA)
- Preliminary data base organization
- Reuse strategy

Develop

Development efforts for Version 1 should concentrate first on delivering the product specified for the version and secondly on establishing a technical base for future development. With a first version that emphasizes scope and content, the development efforts might include:

- Menus and navigation screens (e.g., visual basis pull-down menus in windows)
- Maintenance screens for key data entities
- Entry/processing screens for key business scenarios
- Key results (reports, screens)
- Key system integration features (e.g., access to server from client or access to another application's DB2 data base)
- Menu items and/or screens which validate the breadth of data entities processed

It is important in Version 1 to get as much visibly "done" as possible. This helps establish a good working relationship with the customers. The team must, therefore, be careful about becoming "bogged down" in technical areas. For example, there may be a difficult algorithm or data base design issue that leads to much discussion and consternation. In this situation, look for some simple way to get something done and put the problem on an issue list for the next version.

Even with rapid development, there is *too fast*. Version 1 should be used to establish some development guidelines. With most high-powered development tools, "spaghetti code" or "spaghetti data" can be developed much more quickly that with more traditional languages. While a primary criteria for these tools is that they reduce the effort required to

modify the application, initial set-up can make a big difference in how effective the tools operate. So even in Version 1 the team needs to establish development guidelines such as naming standards, reuse approach, data design considerations, etc. Hopefully the team is experienced with the tools and these guidelines are already in place and only need tailoring to the project.

Deliver

In most cases, Version 1 will not involve much work on delivery/installation tasks. There may be some initial planning for these tasks, for example in many IS applications there is a significant conversion effort. If so, a tentative conversion issues list, maybe not even a full plan, might be started in Version 1. If the final installation is to be across a wide customer base some initial ideas might be generated during Version 1.

Critical Success Factors

The first version is important in getting the project off to a good start. Especially when Evolutionary Development is new, or new to a particular customer, skepticism may exist. Early success in the first version can minimize the skepticism and establish a base for real customer involvement. Critical in this early stage are:

- A completed version with demonstrable functionality
- Validation of key data model characteristics
- Project scope confirmation
- Good team relationships

Version 2: Features, Data, and Visual Interface

Objective

The objective of Version 2 is to further develop the application into a product which meets the basic feature set criteria of the customer. As discussed under process description, several "sub-versions" may be required to develop the full feature set. The development team should strive to meet the customer's product specifications within the overall established constraints (e.g., time) of the project.

Deliverables

- Product Version 2
- Revised Documentation
- Updated Evolutionary Version Plan
- Customer Focus Group Results

Version 2 Process Description

Versions 1 through 4 are used in this guide as examples of the content of products as they evolve. In practice, projects of longer than 3-4 months will usually have more than 4 versions. One of the usual ways of "versioning" products will be to develop different feature sets in each version. The table shows an example of how an application being developed in six versions might be planned. In the table:

- S -- feature's basic **scope** is developed
- C -- feature's main **content** is developed
- D -- feature's **details**, e.g., error handling or help screens, are developed
- R -- a feature is **reworked** based on Customer Focus Group feedback

Feature Set	V1	V2	V3	V4	V5	V6
Order entry	S	C/R	D/R	R		
Order pricing	S	C/R	D/R	R		
Warehouse picking	S		C/R	D/R	R	
Shipping	S			C/R		R

Table - Features by Version Example

One method of prioritizing features with customers is to state up-front that any feature scheduled for Version 4 implementation has a reasonable probability of "slipping" off the schedule if tradeoffs are necessary. This begins a continuous process of weeding out "specification creep" and "gold-plating specifications." While weeding out specifications that may be marginal and often time consuming to implement, the team should continue to look for easier-to-implement abilities that offer "something extra" to the customer.

Specify

The specification process delivers any revisions needed to the documentation from Version 1. Corrections may be required to features or data as a result of the first Customer Focus Group session. The team should review the change forms completed from the Customer Focus Group as one of the inputs to this effort. It would be normal at this juncture to put additional effort into the data modeling effort. Often the Customer Focus Group from Version 1 identifies areas in the data model that are either more complex, or very different from the original thoughts. After the customers have seen the initial version via the Customer Focus Group, they may have more enthusiasm for assisting with the data analysis also.

During Version 2, additional specifications may be required for additional functionality not covered in Version 1. Often, complex functionality can be postponed until Version 2 and then sub-teams can concentrate on further analyzing the customer's needs in these areas. Often this functionality may involve complex calculations or algorithms that were "stubbed" in Version 1. If the needs are broad enough, another 1/2 to 1 day mini-JAD session can be convened. In many cases the customer assigned to the project can gather most of the information through personal knowledge or more traditional interviewing techniques.

Design

While Version 1 addressed overall design issues, design issues addressed during Version 2 are more detailed. At this stage developers will know a great deal more about design issues from building the first version. The Project Team should be willing at this point to abandon much of the prior coding if new design alternatives seem to provide performance enhancements. Although features will change, design issues such as physical data base design, major event/version structures, and external interface handling should begin to solidify. Decisions about functionality assignment to client or server components of the system, and data

distribution issues such as replication/concurrent update, should be stable by the end of this version.

Develop

Development efforts for Version 2 begin to add both functional and technical depth to the product. Additional effort should be directed towards the user interface and defect removal so customers can actually sit at the keyboard during the next Customer Focus Group. The following are enhancements for Version 2 development:

- Menus and navigation screens have reasonably full range of functions and data in the system
- Improved stability over Version 1
- Maintenance screens for most data entities and code tables
- Entry/processing screens for business scenarios
- Expanded results (reports, screens) should put additional effort into formatting
- Key system integration features (e.g., access to server from client or access to another application's DB2 data base)
- Screens which validate the breadth of data entities processed
- Sample "help" facility screens
- Test plans, cases, and data should be organized and partially implemented

Deliver

During Version 2 the Project Team should begin serious thinking about delivery/installation issues. As the number of potential users rise (from a few for some IS applications, to many millions for some commercial products), advanced planning for delivery needs to be addressed in earlier versions.

Critical Success Factors

- Convergence of developer and customer viewpoints
- Validation of most of the data model
- Reasonable progress on feature set implementation
- Key design decisions made

Version 3: Performance and Quality

Objective

The objective of Version 3 is to insure that performance specifications are fully developed and implemented and that quality related to defect levels or "stability" is approaching acceptable levels. As with other versions, additional features or revisions to prior features are an integral part.

Deliverables

- Product Version 3
- Revised Documentation
- Updated Evolutionary Version Plan

- Stability and performance testing results
- Customer Focus Group Results

Version 3 Process Description

The basic categories for version 3's processes are similar to those for versions 1 and 2, but with emphasis on different activities.

Specify

The major activities done for Version 3 include:

- Revising all features requested in the Version 2 Customer Focus Group changes
- Developing/finalizing specifications for inter-system interfaces
- Developing specifications for business and systems error handling and security
- Finalizing performance attribute specifications

Particularly in business applications, Project Teams often get bogged down in error-handling routines and miss the "main-line" flow. The main-line is defined as the execution of business processes as they are intended to be performed if Murphy is asleep (e.g., no errors are encountered). Versions 1 and 2 should concentrate on getting the main-line correct. In Version 3 the Project Team should concentrate on how the application might fail, from either a business or technical perspective. The specifications can then be developed on how to counter act the failures.

Design

By Version 3, any reuse strategy put in place by the developers and architect should be implemented. Any common modules, rules, data base procedures, etc. should be in place.

By Version 3, most system-level design should be finished. Version 3 design efforts will concentrate more at the sub-component level or in areas that may not have been implemented before.

Develop

Development efforts during Version 3 will concentrate in three main areas:

Implementing features specified for Version 3 and changes from the previous Customer Focus Groups. Visual interface changes should not be significant at this stage.

- Performance enhancements to meet attribute tolerances specified. This would include tasks such as tuning data base performance and stress testing the network.
- Quality testing. Testing should become much more through during Version 3. Customer testing scenarios should become much more detailed. Developers should have additional test cases in place to test the full range of the product.

Deliver

During Version 3 the Project Team should finalize thinking about delivery/ installation issues. Conversion programs, installation routines,

training programs, final quality assurance plans should all be at an advanced stage of completion.

Critical Success Factors

- Convergence of visual interface approach
- Satisfactory performance testing
- Relatively "bug free" Customer Focus Group results
- Key implementation and conversion decisions made

Version 4: Total Product Components

The objective of Version 4 is to produce a total product including all product and service components. Some of these components, e.g., user documentation, may have been planned or even started prior to Version 4, but final development and testing are accomplished here.

Deliverables

- Product Version 4
- Completed Documentation
- All Product Components
- Customer Focus Group Results

Version 4 Process Description

There will be a wide range of activities in this version depending on the type of product being delivered. A commercial software product will have extensive efforts for such items as user documentation, possible international localization, and refinement of installation procedures. "In-house" software products may have many of these same activities, but some small applications may have minimal effort on these items. Some of the components which might be developed or finalized during Version 4 include:

- Product Components
  - Help screens
  - On-line training material
  - Installation software
  - Conversion software
  - Operations set-up
  - Security set-up
- Services
  - Training
  - Help desk
  - Reference Material
  - User Guides
  - Operations Guides

Some of these items may be extensive and therefore need to be started in previous versions. For example, User Guides may need to be "versioned" also, with preliminary outlines and material examples being developed in Version 3 (or even 2). In general, any part of the product or service that

takes a significant effort or is subject to change with the software development should be considered as a candidate for "evolutionary development."

Critical Success Factors

- Completion of all Product Components
- Customer acceptance of final Customer Focus Group results

Quality Assurance and Release Preparation

The objective of this phase is to provide final assurance to the customer that the product is ready for installation.

Deliverables

- All Product Components ready for final installation

Process Description

This phase will vary substantially from organization to organization and even within an organization for different size projects. Tasks done during this phase might include:

- Beta testing
- Final integration and stress testing
- Final preparation and printing of all documentation
- Final preparation of training material
- Final release to conversion or manufacturing

Deliverable (By Phase and Version)	Team Roles						
	Project Sponsor	Project Manager	Architect	IT Analyst	Business Process Expert	Developer	Tester
Justify Phase							
Concept Paper	A	D		D/P	D/P		
Project Charter	A	D		I	I	I	I
Project Data Sheet (PDS)	A	D		D/P	D/P		
Feasibility Study Report	A	D	D/P	D/P	D/P	I	
Service Request	D/P	D/P	I	I	I	I	I
Needs Assessment Results	I	I					
"Ability-To" Statements	I	I					
Contracting Documents (RFI/RFP)	A	D		D/P	D/P		
Prepare Phase							
Project Structure	A	D					
Resource Assignments		D					
Application Segmentation Plan	A	D	D/P		D/P		
Version Delivery Plan	A	D	D/P	D/P	D/P		
Project Plan	A	D	D/P	D/P	D/P		
Configuration Management Plan		D	D/P				
Contract/Stmt of Work with Vendors	A	D					
Version 1 - Concept							
Business Context Model	I	I		D	D/P		I

Business Process Model	I	I		D	D/P		I
Data Flow Diagrams		I		D	D/P		I
Screen Prototypes				D		D/P	I
Logical Data Model		I	D/P	D	D/P		I
"Ability-To" Statements		D/P		D/P	D/P		I
Screen/Report Layouts		I		D	D/P	D/P	I
Screen Navigation Maps		I		D		D/P	I
Client/Server Strategy	I	I	D	D/P		D/P	I
System Architecture		I	D	D/P		D/P	I
Version 2: Features							
Object Models		I	D/P				I
Physical Data Models		I	D/P	D/P		D/P	I
Business Rules	I	I	I	D/P	D/P	I	I
Program Specifications		I	I	D/P		D/P	I
Database Schema		I		D/P		D/P	I
Physical Data Dictionary		I	D/P	D/P		D/P	I
Version 3: Performance							
Data Conversion Specs		I		D/P		D/P	I
Data Edit Rules		I	D/P	D/P	D/P		I
Error Handling Specs		I		D/P	D/P	D/P	I
Security Interface Specs		I	D/P	D/P		D/P	I
On-line Help Specs		I		D/P	D/P		I

User Documentation Plan	I	I			D/P		
System Test Plan		I					D/P
Integration Test Plan		I					D/P
Version 4: Total Product		I					
User Training Plan		I			D/P		I
User Training Materials		I			D/P		I
Help screens		I		D/P	D/P	D/P	I
Operations Manual		I		D/P			I
Network Manual		I		D/P			I
Performance/Stress Test Plan		I					D/P
Fault Tolerance Test Plan							D/P
System Test Journal				D/P		D/P	D/P
Beta Test Plan				D/P			D/P
Beta Test Journal						D/P	D/P
Transition Plan				D/P	D/P	D/P	I
Data Conversion Plan			D/P	D/P	D/P		I
Data Population Plan			I	D/P	D/P		I
Hardware Installation Plan			D/P				I
Software Installation Plan			D/P				I
Version 5: Final Release							
User Manual	I	I		D/P	D/P		
User Training Plan	I	I			D/P		

User Acceptance Test Plan	I	I		D/P	D/P		D/P
Converted Databases		I			D/P	D/P	I
Populated Tables		I			D/P	D/P	I
Installed Code	I	I		D/P		D/P	I
Acceptance Test Journal		I					D/P
Evaluate Phase							
Project Retrospective	I	D/P	D/P	D/P	ID/P	D/P	D/P
Project Files	I	D/P	D/P	D/P	D/P	D/P	D/P
Post Implementation Evaluation	I	D/P	I	D/P	D/P	I	I
Project Estimation Guidelines		D/P	D/P	D/P		I	I

Team Role Responsibilities: *"A" - Approval, "D" - Develop, "P" - Participate, "I" - Information Only*