



# RADXP

---

METHODOLOGY FOR DEVELOPING  
SERVICE ORIENTED APPLICATIONS

# RADXP

## METHODOLOGY FOR DEVELOPING SERVICE ORIENTED APPLICATIONS

---

### INTRODUCTION

---

RADXP is a rapid application development methodology for solutions delivery. RADXP is a focused application of the Rational Unified Process (RUP)<sup>1</sup> and Extreme Programming (XP) to the challenge of delivering webservices. This methodology enables solutions providers and their customers to control time, scope and costs. The result is a superior solution completed within 90 days.

The RADXP methodology divides a solution delivery into five phases. Between each phase is a firebreak, a point where the solutions provider provides the customer a deliverable for the customer's approval. For instance, at the end of the first phase, the customer receives a requirements specification for its approval.

	<b>Phase</b>	<b>Duration</b>	<b>Deliverable</b>
1	Requirements	1 week	Requirements specification
2	Architecture	1 week	Solution architecture specification
3	Design	2 weeks	Solution designs, models, and mockups
4	Execution	6 weeks	Product
5	Production	2 weeks	Deployed solution

*Figure 1 RADXP methodology has five phases*

Since 1994, solution development has undergone dramatic change to what we now call webservices development. Figure 2 contrasts some of these changes. Jeffrey Ricker and Paul Lyman developed the RADXP methodology in response to these changes. The RADXP methodology enables solutions providers and their customers to more than cope but actually excel in this new development environment.

<b>Traditional development</b>	<b>Webservices development</b>
Compiled and monolithic	Component-based and distributed
Single language such as C++	Multiple languages such as Java, .NET, JavaScript, HTML and XML
Single integrated development environment (IDE)	Multiple development tools for page layout, graphic design, programming, etc.
Set releases	Continuously evolving
Contributors are programmers or developers	Contributors come from multiple disciplines such as graphics and marketing
Homogenous and tangible	Heterogeneous and ethereal
Single platform	Multiple platforms
Ported to different operating systems	Ported to different application servers
Contributors co-located	Contributors from any location
Software distinctly separated from content	Distinction between content publishing and

---

<sup>1</sup> RUP is a registered trade mark of IBM

publishing	software development is blurred
Simple text or widget interface	Dynamic multimedia user interface
Built for a specific use within a specific environment	Design does not anticipate end use or environment

Figure 2 Comparison between traditional development and webservice development

This paper describes the RADXP methodology, its different phases and deliverables. The RADXP methodology empowers solutions providers and their customers to focus on success. The customer sees tangible capabilities delivered in rapid succession. Just as important, the customer enjoys peace of mind that the solutions provider will consistently deliver what they have committed to deliver, on time and in budget.

---

### THE SOLUTION MANAGEMENT EQUATION

---

In order to manage the delivery of the solution, we must solve three variables simultaneously. Those variables are time, scope and resources. The variables are interdependent. The value of one affects the value of the other. If these variables are not defined or determined, then the solution will either fall short of the customer's expectations or the solution will go over budget. A true solution provider defines both possibilities as a failure. The RADXP is the solution provider's means of working with its customers to effectively solve these three variables simultaneously so that the target comes quickly into focus.

Given these three variables, there is an equation that can be used to roughly estimate the project. It is a simple equation; the scope of your project is directly predicted by the amount of time and resources available. Scope is proportional to resources multiplied by time.

**$S \propto RT$**  *scope is proportional to resources multiplied by time*

- If we increase the scope, then we must increase the time or resources or both.
- If we decrease the time, then we must increase the resources or decrease the scope.
- If we decrease the resources, then we must increase the time or decrease the scope.

There are realistic boundaries to this equation. For instance, we cannot employ infinite resources or zero time. While resources can sometimes be scarce, the bigger factor to be considered is that of utilization. How many people can realistically work on creating a solution? There arises a need for coordination of effort whenever more than one person works on a project. The more people that work on a project, the more coordination required. In most efforts, there is a point where adding more people to the project is counterproductive. The effort required to coordinate the new people into the project actually exceeds the effort the new people contribute to the project. One famous engineer summed up this dilemma with the following analogy: While it is possible for one woman to have one baby in nine months, it is impossible for nine women to have one baby in one month.

*The “Pregnant Woman” Rule: “There is a point where adding more people to the project is counterproductive. The effort required to coordinate the new people into the project actually exceeds the effort the new people contribute to the project.”*

Another limit is perfection. Given enough time, engineers can usually come up with the perfect solution, from a technical point of view. However, if the perfect solution costs \$2 million but only \$300 thousand is available, then it really isn't the perfect solution. In other words, the scope must remain within the bounds of cost. General Electric has achieved great success releasing quality solutions with the mantra "Good enough."

*The Cost Rule: “The perfect solution is only the perfect solution if it is affordable.”*

Only two of the components values may be held fixed. If you have a given number of resources and must deliver on a given date, the scope is your result. Similarly, if you have a fixed scope and time, the result will be the number of resources needed. Keep in mind that you may not be able to balance the equation. If this happens, then concessions must be made. If the project, given fixed resources and scope, takes longer than you are willing to wait you must decrease the scope (read the number of features or functions that are included), increase resources (if that is a plausible alternative), or be prepared to wait anyway.

Since people account for most if not all of the resources required to build a solution, the scope of a solution is proportional to the number of people working on the problem and the amount of time those people have to work on the problem. People multiplied by time define a cost, so the scope of a solution is directly proportional to the cost of the solution. This is the Iron Rule of project management.

***S  $\propto$  cost** The Iron Rule: “Scope is proportional to cost”*

The following chart illustrates the equation graphically. The overlapping zones indicate a solution based on the adjustable variable. The center zone indicates that all variables are within bounds and the equation is balanced. Keep in mind that if the scope, resource and time variables are unbalanced enough, the result is a graph in which none of the circles touched each other.

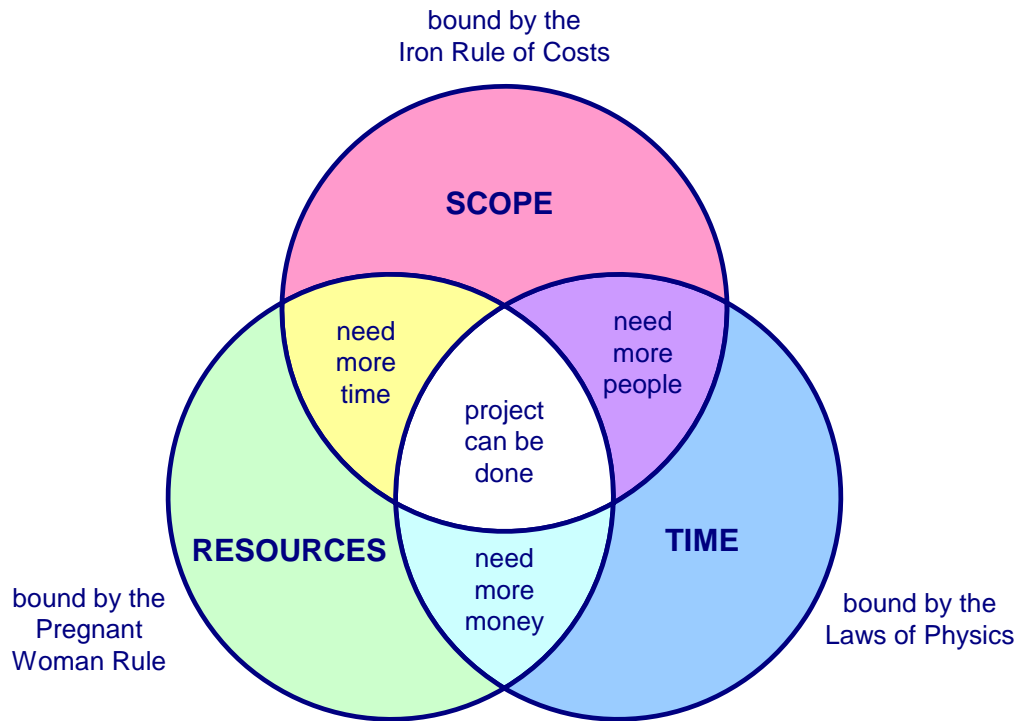


Figure 3 Visually solving the three variables of solution management

#### CONTROLLING TIME

The RADXP methodology helps the solution provider solve the scope-resource-time equation by fixing the time variable. RADXP works in a 90-day delivery cycle. The development team delivers something functional and tangible every 90 days.

A 90-day cycle is practical for many reasons. Corporations are accustomed to 90-day reporting cycles. The period is not too long or too short. The period must be less than 6 months because the environment, especially technology, changes faster than 6 months. The period must be longer than 6 weeks or too much time is spent ramping up and ramping down. The team would never get in stride.

	<b>Phase</b>	<b>Duration</b>	<b>Deliverable</b>
1	Requirements	1 week	Requirements specification
2	Architecture	1 week	Solution architecture specification
3	Design	2 weeks	Solution designs, models, and mockups
4	Execution	6 weeks	Product
5	Production	2 weeks	Deployed solution

Figure 4 RADXP methodology executes five phases in 12 weeks

#### CONTROLLING SCOPE

The RADXP process helps control scope. By the end of the design phase, the solution provider and the customer have good documentation of what needs to be built during the execution phase. This documentation helps prevent expanding scope, or *mission creep*, as it is sometimes called.

Projects that run 12 months or longer are never done. Before the project is half way done, the requirements have changed. Some solutions by their nature cannot be completely achieved in 90 days. The RADXP methodology forces the customer and the development team to focus on true priorities and do first things first. By analogy, look at the U.S. space program. The United States did not go directly to the moon. First, we launched a satellite, then a dog, then a man, then the man made a space walk, etc. A billing or logistics system should be no different. Even if one could launch everything at once, the users and the business processes could not handle it. . A staggered implementation naturally takes into account the inherent problems of learning curves, transitional efforts, undetermined process definition and provides flexibility of design.

### CONTROLLING COSTS

Our RADXP methodology controls cost by accurately accounting for costs before they are incurred. The experience of the solution provider's project managers enables them to reliably estimate the cost, scope and time of most solutions. However reliable they may be, though, those figures are still merely estimates.

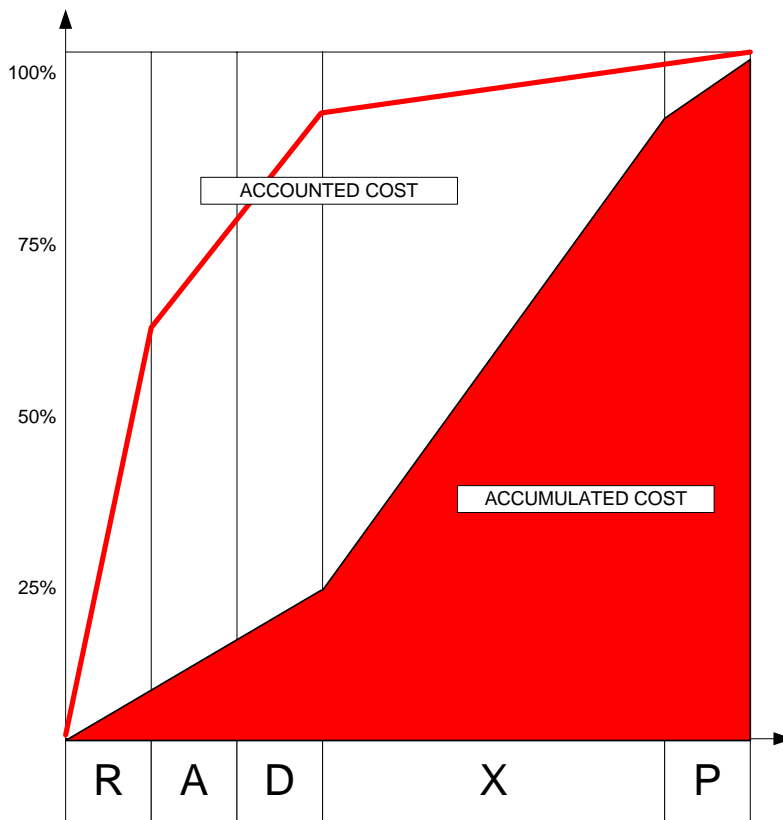


Figure 5 RADXP enables cost control

Every senior engineer learns through experience: you don't know what you don't know. During the requirements, architecture and design phases of our methodology, we gain understanding of the true scope and challenges of the project. At the end of the requirements phase, plus or minus 40% of the eventual costs are accounted for. By the end of the architecture phase, plus or minus 20% of the eventual costs are accounted for. By the end of the design phase, we know within plus or minus 10% of what the eventual costs are. Yet, by the end of the design phase, we have only incurred 25% of the

total cost of creating the solution. Very little cost is incurred before most of the eventual costs are accounted for.

**FOCUS ON SUCCESS**

The RADXP methodology empowers solutions providers and their customers to focus on success. The customer sees tangible capabilities delivered in rapid succession. Just as important, the customer enjoys peace of mind that the solutions provider will consistently deliver what we have committed to deliver, on time and in budget.

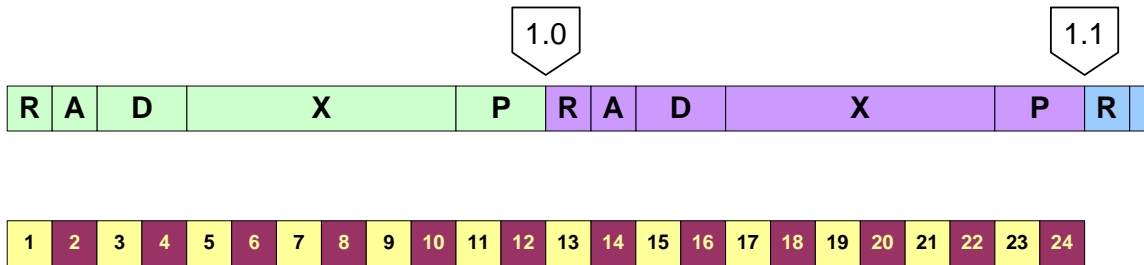
**DEVELOPMENT CYCLES**

**SEQUENTIAL VERSUS PARALLEL**

Depending on the scope of the solution and the expectations of the customer, the solutions provider will employ the RADXP development cycle sequentially or in parallel. For one-off solutions and smaller projects, the RADXP development cycle moves sequentially. One full development cycle is completed before another is begun. In this mode, planning phases are completed in 4 weeks, development is 6 weeks and production is 2 weeks. The complete cycle is 12 weeks, or 90 days.

	<b>Sequential mode</b>	<b>Parallel mode</b>
Requirements	1 week	2 weeks
Architecture	1	2
Design	2	2
Execution	6	9
Production	2	3
<i>Total</i>	12 weeks	18 weeks

*Figure 6 The length of each phase of the development cycle*



*Figure 7 Sequential 90-day delivery cycle*

For larger solutions and commercial product releases, the RADXP development cycle operates in parallel. Four weeks into the execution phase of the current release, the management team begins the planning phases (RAD) for the next release. By working in parallel, the process phases can extend while still maintaining a 90-day delivery cycle. The delivery cycle in parallel mode takes a total of 18 weeks (135 days), but since the planning phases occur in parallel, a release occurs every 12 weeks. This approach only works in larger projects with at least some team members dedicated to planning.

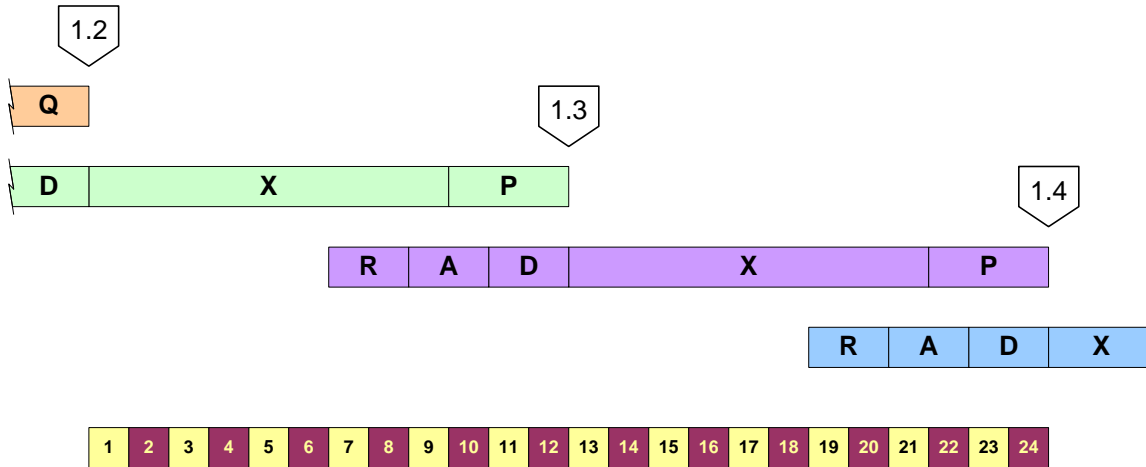


Figure 8 Reiterative (parallel) 90-day delivery cycle

From the customer's perspective, there are many reasons for rapid, iterative development cycle.

- **TRANSITIONAL EFFORT.** People are creatures of habit and must be transitioned from the 'old way' to the 'new way' in steps in order for the new process to fully take hold with as little interruption as possible.
- **UNDETERMINED PROCESS DEFINITION.** It's the standard fare of "I'm not sure how to solve our problem" or "I'm not sure what we need". Staggered implementation allows time to resolve those questions while implementing the obvious core issues immediately.
- **FLEXIBILITY OF DESIGN.** As the process definition evolves, requirements change. By focusing on small implementation steps in 12 week cycles, starting with the most fundamental steps first, the "frills", could have and nice to have can be fleshed out and implemented in later cycles without undue penalty.

### RHYTHM

Every organism lives with an innate sinusoidal rhythm: work and rest, hunt and hibernate, bloom and fade. Organizations also have a natural rhythm. In the United States, that rhythm is based on a quarterly and yearly cycle. Corporations must deliver every quarter. The RADXP requirements and release cycle meets the natural rhythm of corporate delivery expectations and business change.

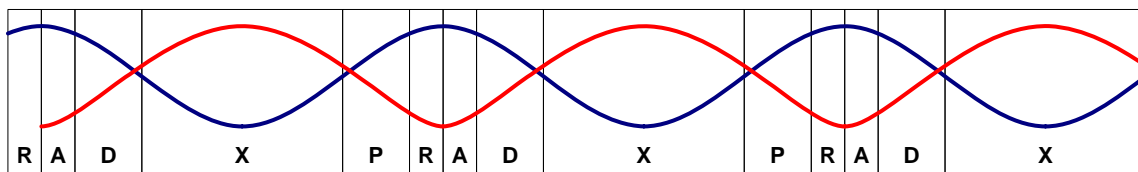


Figure 9 RADXP accommodates the natural rhythm of an organization

RADXP also accommodates the needs of the individual managers and developers. People cannot give 110% all of the time. Periods of relative calm must follow periods of intense effort, or teams become burned out. RADXP allows management to work in a complementary rhythm to the development team. Management works most intently during the planning phases (RAD) and the

production phase (P). Likewise, the development team works most intently during the execution phase (X). The result is a sustained level of intense effort that facilitates individual rest.

---

## REQUIREMENTS PHASE

---

### PURPOSE

In the first phase, the solutions provider works with the customer to define and document the solution requirements.

- Define the precise business purpose of the application. Why does it exist? What should it be accomplishing, providing, or allowing?
- Define the purpose of each subsection if the application already exists. For a new application, group the purposes logically. Defining the purpose is important because you need to know if you have achieved your objective when you are done.
- Prioritize features as to which are essential, which are nice to have and which are not necessary. The purpose of the application is the means of deciding priority.
- If possible, you identify metrics of success that are immediately apparent. One possible metric is number of users. Metrics should not be forced and should not add any significant workload.

### USER OR CUSTOMER EXPECTATIONS

Define the user or customer of the application. There will usually be more than one type of customer. That is, different people will be using the application for different reasons, with different expectations and possibly to achieve different goals.

For complex or business critical applications, create scenarios or fictitious sample profiles of each of these customer types. Some simple examples of customer profiles are:

- “The CEO wants to know the number of inbound trucks at any given time. That number is his "gut check". He doesn't want to go through a lot of screens to get this information. He travels a lot, so the information needs to be accessible to him outside of the office.”
- “Sue is the shipping manager. When the COO calls her, she needs to be able to access specific details about ship set status immediately.”

Defining customers helps decide what should be included in the application and prioritizing features. Defining customers is also important in creating the navigation of the site.

### CUSTOMER PLATFORM

Define the limits and capabilities of the user's machine.

- Is there a standard hardware?
- Is there a standard browser brand and version?

- What is the pipe size: T-1 or 28.8? Is access LAN, WAN or dial-up?
- Are there extraordinary firewall restrictions?

Defining the customer platform specifies the limits to the capabilities you will design in the product. For instance, you can't plan on using an applet if customers are using Netscape Navigator version 2.

### **COMPETITORS**

What internal or external competitors are there to this application?

A competing application is any application that is attempting to achieve or is capable of achieving the same purpose. Some competitors are older existing applications that do not have the full capability required. For instance, Lotus Notes may have been previously used unsuccessfully to try to achieve the same objective. Another division within the company may have built an application that does similar things. There may exist a company that provides this service for a fee.

Each competitor must be addressed and reason established why this application is more advantageous for the user and the company.

Defining competitors helps to define the priorities of the application. For instance, suppose you decided five different sections were required to achieve the business objective. One section is currently done in a Lotus Notes database, though not to your customer's expectations. The purpose of another section is mostly achieved by another company's website, but it is hard to use and doesn't have all the information necessary. You can focus your initial development on the other three sections first. You can give directions on where to find the older Lotus Notes app and the outside website as a temporary solution until you are able to replace them. In short, it allows the focus to be given to the most critical areas first

Defining competitors can also save time, money and grief.

### **BRAINSTORM AND PRIORITIZE**

Brainstorm all the possible things that could be included in the solution. Group these possibilities into

- Must have
- Should have
- Could have

Prioritize the "must have" items based on purpose, customer and competition.

You may also rate each item based on technical difficulty and maintenance difficulty. Use simple ratings such as "easy", "hard" and "impossible".

Review the list and its prioritization with the team. You can do this brainstorm and review one of two ways, depending on your business culture:

- 1) One or two people conduct interviews with the team members informally, asking them what they think should be or could be included. The interviewers prioritize the list and present their list to the team for discussion in a half-hour meeting.
- 2) The entire team assembles and has a brainstorming session. The team then prioritizes the list. This meeting would take one to four hours.

It is important that those participating in the prioritization know the purpose, customer and competition before hand. Distribution of these items a day or two in advance of the meeting will give team members time to review and reflect. This will help the meeting to focus on the tasks at hand and limit the time necessary to get everyone “up to speed”.

#### **DELIVERABLE**

At the end of the requirements phase, the solutions provider delivers to the customer a Solutions Requirements Specification that contains:

- The business objective with possible metrics of success
- User profiles
- Environmental limitations such as bandwidth, security and client platforms
- Existing alternative solutions
- A prioritized list of features

The customer must sign off on the requirements document for the solutions provider to proceed. The requirements phase necessitates intensive customer involvement to be successful. The whole requirements phase should only take a week if the necessary individuals within the customer organization are available.

---

### **ARCHITECTURE PHASE**

---

#### **SYSTEM ARCHITECTURE**

Application architecture is the physical environment of the solution and its layout. The application architecture should include the following:

- Hardware platforms and their operating systems
- Network topology, including domain names, IP addresses, etc.
- Firewalls and software security regimes
- Databases and application servers
- Legacy data sources
- Message oriented middleware and other communications software

- Points of contact and personnel assigned to network security or IT

### **APPLICATION ARCHITECTURE**

The application architecture defines the major components or subsystems of the solution and how they will interrelate. In particular, the application architecture defines the information flow within the solution and between the solution and existing systems.

Often the components of the application architecture are dictated by what the customer already owns.

### **OPEN SOURCE ALTERNATIVES**

Based on the initial requirements, the solution provider conducts a search of existing open source initiatives. The objective is to find open source software that fulfills part or parts of the solution requirements. The software is assessed for:

- **CAPABILITY.** Does the software have some or all of the features we need? How stable is the software? How much work is required to make it stable?
- **INTEGRATION.** Can the solution integrate the open source software into its architecture? Does the solution provider have the skills necessary to work with the open source software?
- **COOPERATION.** Does the particular open source license structure meet the business requirements? Is the direction of the project (its roadmap) compatible with the customer?

The solution provider presents these options to the customer with the associated risk. The decision must include (a) whether to employ the open source software and, if so, (b) whether to contribute back to the open source project.

### **NAVIGATION ARCHITECTURE**

Create a navigation diagram of the site showing sections and subsections. The diagram can be a tree or an outline created in text, image, or special file format.

The navigation diagram is a conceptual document. It doesn't have to include every individual web page or screen in the application. Since it is hyper linked, an item isn't restricted to being under only one section. The diagram should consider expansion to include the "should have" and "could have" items on the brainstorming list in the definition phase. Make sure to document as thoroughly as possible pages, page flow and any upgrade plans or ideas as the details evolve. This will document design justification and prepare or anticipate additional functionality upgrades with minimal effort.

### **IDENTIFY CHANNELS**

Channels are built around information that can be pushed to customers on their request. They are usually areas of the application that are updated regularly with important or time-sensitive information. Identifying channels is important in structuring the application. For some push technology, the channel components may need to be in special directories. Careful consideration of channels is also important in designing application maintenance.

## IDENTIFY APPS

Identify inter-activity in the application such as forms and database queries. Identify the databases or other external assets the application will need to interface with.

Get team feedback and consensus on the diagram. Make sure that multiple customers and disciplines are considered as needs, wants and skill levels vary greatly.

Navigation is important because poor navigation can kill an application. If customers cannot find what they are looking for, they will probably not come back, they probably won't contribute, and thus your application will fail to meet its business objectives. The same holds just as true for internal intranet customers as it does for extranet and website customers.

### **DELIVERABLE**

At the end of the architecture phase, the solutions provider delivers to the customer a Solutions Architecture Specification that contains:

- System architecture
- Application architecture
- Open source alternatives
- Navigation architecture

The customer must sign off on the architecture document for the solution provider to proceed. The architecture phase requires less customer involvement than the requirements phase. The whole architecture phase should only take a week if the necessary individuals within the customer organization are available.

---

## **DESIGN PHASE**

---

### **SCHEMA DESIGN**

During this phase, developers design the primary data structures that the application will use. These structures include database table structures and XML message structures. The developers need to assure that they have the schema and metadata for any other legacy data structures the solution may employ.

In this phase programmers also create a skeleton of the class libraries they will develop. The designs will include object drawings that identify the primary components and interfaces of the code to be created. This task is necessary to provide guidance for developer cooperation. It is not intended to be a thorough and rigorous specification of every object in the solution. The developers need only design the primary interfaces of the primary components.

### **SCREEN DESIGN**

The design phase is the last of three planning phases. Here the solution begins to take on tangible form to which the customer can truly relate.

In the architecture phase, we created a wire diagram identifying the screens of the application. In the design phase, we create mock-ups of these different screens. This step is usually the most

important with respect to involvement by the customer. In this step, the customer sees how he or she will interact with the application.

The screen designs specify how the navigation is implemented, the layout of tool bars and editors, color schemes, icons, etc. Not every screen is created. Rather, we create enough sample screens for the customer to make a decision on look-and-feel.

It is very important to get the customer's consensus on the screen design. Creating the screens is often the most labor-intensive task in the execution phase. This process of building a "mock up" allows the discovery of shortcomings, problems and difficulties prior to beginning the execution phase. Changes made to the screen design during the execution phase will directly impact the delivery price and schedule.

### **DELIVERABLES**

At the end of the design phase, the solution provider delivers to the customer a Solutions Design Specification that contains:

- User interface (screen) designs
- XML and other message type schema
- Database schema
- Object interface design

The customer must sign off on the design document for the solution provider to proceed. The design phase approval is the most important approval step. Changes made to the design after the design phase will directly impact the delivery cost and schedule. The whole design phase should only take two weeks if the necessary individuals within the customer organization are available.

---

## **NON-PLANNING PHASES**

---

### **EXECUTION PHASE**

In the execution phase, the solution provider developers work rapidly to bring the solution to reality. During the planning phases, the developers have already begun to form in their minds the code that they need to build. Responsibilities for task completion will be divided amongst the developers, enabling them to cooperate effectively, avoiding rework and redundant work.

The previous phases have provided the developers the knowledge they need to fulfill the customer's expectations. It is never possible to anticipate everything, however. Some questions will inevitably arise. It is important that key customer personnel be available to answer developer questions. Rapid response to those questions will ensure a successful release.

### **PRODUCTION PHASE**

Production encompasses testing and deployment. During the execution phase, the solution provider's developers conduct what is called unit testing and white-box testing while they build. During this last phase, testing is more rigorous and formal. Quality assurance is the check that the result matches the design. Execution should follow the plan laid out during the design phase and the quality phase ensures that it does. The testing in this phase focuses on three main objectives:

- Does each component meet its requirements?
- Do the components interact with each other per design and expectation?
- Does the overall product meet specification?

The deployment process is more than just making the application accessible. Deployment is about knowledge transfer. The customer has to understand the solution and how to make best use of it in business operations. The solution provider believes in making customers as self-sufficient as they wish to be. We can assume full responsibility of operation; enable the customer to operate entirely self-sufficiently; or provide a support level anywhere in between.

The quality assurance phase ends with a customer debriefing and satisfaction survey. The debriefing details what the solution provider built and why. The satisfaction survey is a set of ten simple questions ranking one to five. The answers are saved to identify shortcomings in our organization, method or practice.

#### **FOLLOW UP**

Six weeks after the project is completed, a manager who did not work on the project will follow up with the client to ask ten simple questions. The follow up provides the customer the opportunity to point out any shortcomings in the solution that may have arisen.

---

#### **CONCLUSION**

---

RADXP is a methodology for delivering webservices solutions. It is based upon eight years of experience in delivery large-scale web-based applications to government and industry. It is also based upon the Rational Unified Process (RUP) and Extreme Programming (XP). By keeping the time variable fixed, the solutions provider is able to control the cost and scope variables of the effort. The result is rapid, repeatable and replicable success in delivery solutions on time, to expectations and to budget.

---

**APPENDIX A: SUMMARY OF DELIVERABLES**

---

	<b>Phase</b>	<b>Deliverables</b>
<b>R</b>	Requirements	Requirements Specification <ul style="list-style-type: none"><li>• The business objective with possible metrics of success</li><li>• User profiles</li><li>• Environmental limitations such as bandwidth, security and client platforms</li><li>• Existing alternative solutions</li><li>• A prioritized list of features</li></ul>
<b>A</b>	Architecture	Architecture Specification <ul style="list-style-type: none"><li>• System architecture</li><li>• Application architecture</li><li>• Open source alternatives</li><li>• Navigation architecture</li></ul>
<b>D</b>	Design	Design Specification <ul style="list-style-type: none"><li>• User interface (screen) designs</li><li>• XML and other message type schema</li><li>• Database schema</li><li>• Object interface design</li></ul>
<b>X</b>	Execution	Working software
<b>P</b>	Production	Deployed software User training

APPENDIX B: OFFSHORE DEVELOPMENT

