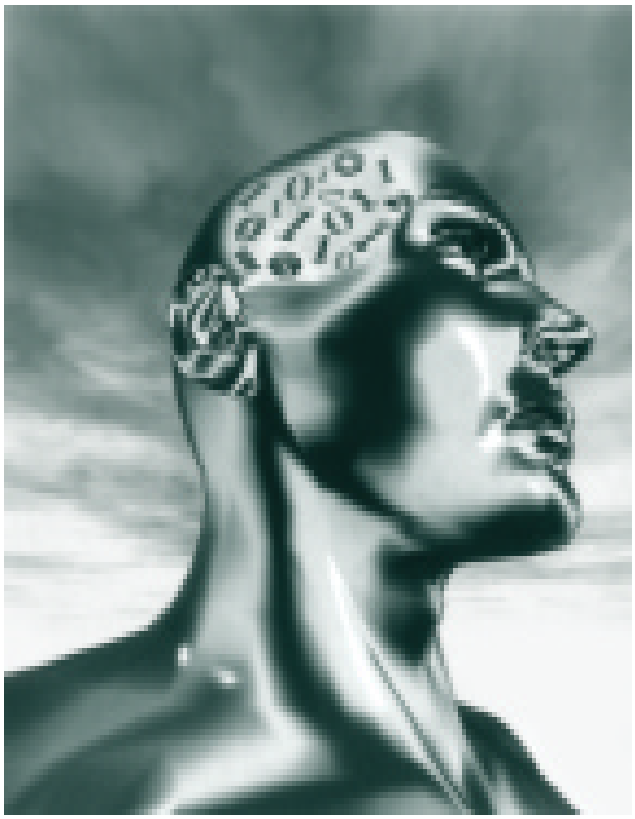
Microsoft
Solutions Framework

Process Model for Application Development

CloverLink Systems Inc. Utilizes the Microsoft Solutions Framework Methodology for Rapid and Efficient Design, Development and Deployment of Applications



CloverLink SYSTEMS, INC.

Your Link to Tomorrow's Business

1486 Sunshine Drive • Glendale, CA • 91208 • www.CloverLink.com
e-mail: Sales@CloverLink.com
800.378.8348

Process Model for Application Development

Contents

Introduction

Microsoft Solutions Framework (MSF) is a framework rather than a methodology that can be adapted to suit the particular needs of an organization. The process model for application development is one aspect of this framework. It describes a lifecycle that can be used for successful software development.

The process model allows a team to respond to customer requests and to change product direction midcourse. It also allows a team to deliver key portions of software faster than would be possible otherwise.

The process model is a flexible component of MSF that has been used successfully in the software industry to improve project control, minimize risk, improve product quality, and increase development speed.

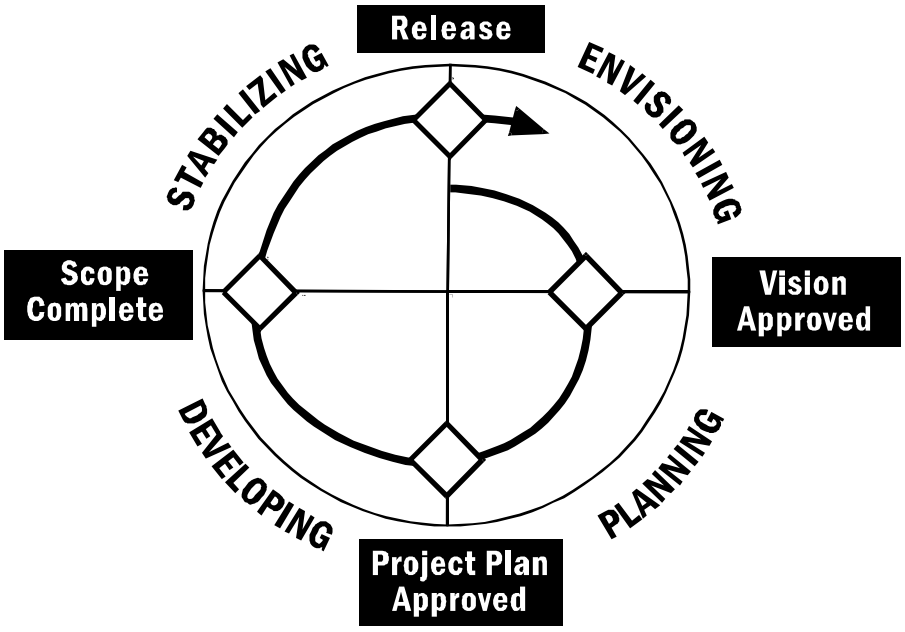
Process Model

Every software development effort goes through a lifecycle, a process that includes all activities in the development cycle that take place up to initial release. The main function of a lifecycle model is to establish the order in which a project specifies, implements, tests, and performs its activities. The appropriate lifecycle model can streamline your project and help ensure that each step moves you closer toward your goal.

Two of the more popular lifecycle models are the traditional waterfall model and the spiral or rapid application development (RAD) model.

In the waterfall model, a project progresses through sequential steps from the initial concept through system testing. This model works well for complex projects in which you can easily specify requirements at the beginning. This model uses milestones as transition and assessment points.

The spiral lifecycle model is a risk-oriented one that divides a software project up into different subprojects. Each subproject addresses one or more major risks until all are identified. The spiral model allows for increased creativity and greater management of risk due to its iterative nature.
The MSF process model combines the traditional waterfall model with the spiral model to use the strengths of both. The process model provides the benefits of milestone-based planning from the waterfall model, as well as the iterative creative process from the spiral model.
The MSF process model is based on milestones. Milestones are review and synchronization points rather than freeze points. They allow the team to adjust the scope of the project to reflect changing customer requirements or to react to risks that may materialize during the course of the project.
Each phase of the development process culminates in an externally visible milestone. These milestones are points in time when all team members synchronize their deliverables with customers and end users; with operations, support, and helpdesk personnel; with the distribution channel (commercial software); and with other key project stakeholders.



Microsoft Solutions Framework (MSF) Process Model

Vision Approved Milestone

The envisioning phase culminates in the vision approved milestone. This first milestone is the point at which the project team and the customer agree on the overall direction for the project, including what the product will and will not include. Envisioning addresses one of the most fundamental needs for project success—unifying the project team.
The team must have a clear vision of what it wants to accomplish for the customer and be able to state it in terms that will motivate the entire team and the customer.
Envisioning, by creating a high-level view of the project's goals and constraints, can serve as an early form of planning and set the stage for the

more formal planning process that will take place during the planning phase of the project.
The deliverables for the phase are:

- Vision document
- Risk assessment document
- Project structure document

Project Plan Approved Milestone

The planning phase culminates in the project plan approved milestone. This second milestone is the point at which the project team, the customer, and key project stakeholders agree on what the project will deliver and design a solution. It also provides an opportunity to establish priorities and set expectations. The project plan approved milestone is essentially the contract with the customer to proceed with the project.
The deliverables are:

- Functional specification
- Risk assessment
- Project schedule

Scope Complete Milestone

The developing phase culminates in the scope complete milestone. At this milestone, all features are complete and the product is ready for external testing and stabilization. This milestone is the opportunity for customers and end users, operations and support personnel, and key project stakeholders to evaluate the product and identify any remaining issues they need to address before it ships.
The deliverables are:

- Frozen functional specification
- Risk management plan
- Source code and executables
- Performance support elements
- Test specification and test cases
- Master project plan and master project schedule

Release Milestone

The release milestone occurs once the team addresses all outstanding issues and ships the product or places it in service. At the release milestone, responsibility for ongoing management and support of the product officially transfers from the project team to operations and support.
The deliverables are:

- Golden release
- Release notes
- Performance support elements

- Test results and testing tools
- Source code and executables
- Project documents
- Milestone review

Process Principles

Overview

The process model provides a key function in project development by specifying which activities the team should perform and when. The model has two other important aspects. One is the close relationship the process model shares with the team model and how both benefit when they are used together. The other is the process model's underlying practices and principles, which include:

- Using versioned releases
- Scheduling for an uncertain future
- Managing trade-offs
- Managing risk
- Maintaining a fixed ship-date mindset
- Breaking large projects into manageable parts
- Performing daily builds
- Using bottom-up estimating

Using Versioned Releases

Microsoft's fundamental product development strategy divides large projects into multiple versioned releases, with no separate product maintenance phase. After the development team establishes a pattern of making good project trade-off decisions and shipping the right products at the right times, it's important to begin cycling through versioned releases as rapidly as possible.

No matter how fast the project team advances, the market, or the technology, or the competition, or the customer's business will advance faster. Versioned releases enable the project team to respond to continuous changes in scope, schedule, and project risk. By frequently updating the product, not only does the development team communicate with the customer, but suggestions for future releases of the product come directly from customer use of the product.

The team delivers a core set of features in its first release and adds features incrementally in later releases until it achieves the full vision for the product. Later versions allow the team to revalidate or update product vision as business requirements change.

Using versioned releases:

- Promotes frequent and honest communication between the team and the customer. The release reflects their best ideas.

- Enables the project team to deliver critical functionality earlier and to obtain feedback from the customer for future releases. When customers know (or sense) that the team will be timely in delivering future product releases, they are much more receptive to deferring features to later releases.
- Forces closure on project issues through the stabilization phase when the project team addresses all issues with the project before release. Using a versioned release allows the team to deal with a manageable number of issues during this phase.
- Sets clear and motivational goals for all team members. The team can easily manage each version's scope and quickly achieve results. This allows team members to see progress rapidly. Their role in determining the schedule means their tasks are not ongoing, but are specific and associated with a tangible result.
- Manages the uncertainty and change in project scope by allowing the team to vary the features and schedule in relation to each other for a specific set of features in an overall technology plan. This allows the team to respond to changes in the business environment for which the solution is being designed. Features that become critical as a result of business changes can be a high priority for the next release. The team starts work on the next release as the current version becomes stable.
- Encourages continuous and incremental feature delivery. Work on a set of new features immediately follows the release of one set of features. As a result, the project team constantly adds value for the user and customer.
- Enables shorter time to market. The project team works with the customer and user to determine which features are the most important and delivers them first. The team schedules other features for release in subsequent versions. The first version's time to market is far shorter than it would have been had the project team decided to implement all of the possible features.

Steve McConnell explains in his book, *Rapid Development*:
"One of the keys to users and customers agreeing to the version-2 approach is that they have some assurance that there will in fact be a version 2. If they fear that the current version will be the last version ever built, they'll try harder to put all their pet features into it. Short release cycles help to build the user's confidence that their favorite feature will eventually make it into the product."
Another way to relieve this condition is to create what Jim McCarthy calls a "multi-release technology plan" in his book *Dynamics of Software Development* where current and future versions of the product are articulated so that the team and customer can trust in the future of the product.

Scheduling for an Uncertain Future

Teams need to address uncertainties in project scheduling and management because the future is inherently uncertain. MSF accounts for future

uncertainties by using two primary approaches: adding buffer time and using risk-driven scheduling.

Buffer time is the equivalent of a military commander holding back reserve troops to account for variations in the results of the attack plan.

Program management determines the buffer time, adding it to the end of the schedule rather than factoring it into the tasks. Buffer time is not an allowance for poorly defined tasks. In almost all cases, having to use the buffer comes at a cost, if nothing more than a required explanation and justification for using the buffer and a plan for correcting the problem in the future.

Because the future is uncertain, a software development team cannot add up all known estimated variables to set a dependable ship date. The team needs to include a buffer to create an internal ship date (based on the summation of the estimated variables) and an external date that includes the buffer.

Chris Peters, former general manager of the Microsoft Word Business Unit, says the following about buffer time:

"The way you do an accurate ship date is that you add buffer time—time which says the future is uncertain, which it clearly and obviously is."

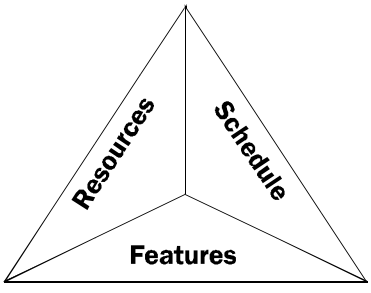
Risk-driven scheduling assigns higher-risk tasks a high priority and includes risk priorities assigned by the customer. If the high-risk tasks require more time than planned, risk-driven scheduling increases the amount of required reaction time. Risk-driven scheduling:

- Encourages early proof-of-concept prototypes.
- Determines which features will be shipped and when.
- Prioritizes tasks based on technical and business risk.
- Lets developers aggressively shoot for the early milestone.
- Signals a warning if the early milestone is missed, pinpointing the need to make adjustments and trade-offs earlier.
- Gives customers greater vision into riskier areas of the project, and manages their expectations in a more productive manner.

Managing Trade-offs

McCarthy states in Dynamics of Software Development:

"As development manager, you're working with only three things: resources (people and money), features (the product and its quality), and the schedule. This triangle of elements is all you work with. There's nothing else to be worked with. And changing one side of the triangle has an impact on at least one other side, usually two."



Project Elements in Triangulated Relationship

The relationship between elements of the triangle tends to be hazy at the beginning of the process. At this point, the team has a rough idea of what it wants to build, an estimate of available resources, and a high-level target delivery date. As the team moves forward through the planning process, the triangle solidifies and the project elements become more distinct. By the time planning is complete, the team should know exactly the nature of available resources, the product features, and the fixed ship date.

It is extremely important to understand that the project variables exist in a triangulated relationship. This concept empowers the team to take corrective action as changes occur during development.

For example, say a given triangle is defined as having 10 resources to deliver 20 features by June 1. During development, the customer discovers a new critical feature that is not part of the original functional specification. Adding this new feature to the triangle creates an imbalance with the other sides. The team must correct this imbalance by dropping features, adding resources, changing the ship date, or some combination of all three actions.

The power of the triangle lies in its simplicity. It is simple enough to draw on a napkin during lunch with a customer to explain the types of tradeoffs that must occur in order to succeed.

Although the triangle is a simple and effective tool, it does not convey the project's priorities with regard to the three variables. One way to document the focus for the project and to manage expectations between the team and customer is to create a trade-off matrix for the project variables and the levels of constraint. This lets the team and customer indicate the manner in which trade-offs should occur.

Project Trade-off Matrix

	Optimize	Constrain	Accept
Resources			
Ship Date			
Resources			

Working together, the team and customer place a check mark in the appropriate column for each of the project variables. The columns are defined as:

- Optimize. To optimize cost/resources is to seek the lowest possible allocation of cost/resources (minimum cost strategy). To optimize the ship date is to set the ship date as soon as possible (early-to-market strategy). To optimize the scope is to ship as many features in the product as possible (maximum benefit strategy).
- Constrain. To constrain the cost/resources is to take a not-to-exceed strategy. To constrain the ship date is to time-box the project. To constrain the features (scope) is to ship at least the essential set of functionality.
- Accept. To accept the cost/resources is to consider a time and materials strategy. To accept the ship date is to acknowledge that the product won't ship before its time. To accept the features (scope) is to achieve one or more of the other project trade-offs by dropping features immediately prior to the ship date.

A team should use the project trade-off matrix as a reference when making decisions. The matrix is not intended to show absolute priorities; it is merely a tool to facilitate communication and understanding. Most important for the project team is that the matrix shows areas in which the customer is willing to compromise. Make sure that no row or column in the project trade-off matrix has more than one check mark. Any other combination poses serious risk to the project and must be accounted for explicitly in the risk management plan.

In order for a team to be successful, at least one check mark must be in the "accept" column. This means that the team owns one side of the triangle (that is, owns at least one variable) so that the team is empowered to manage change and risk, and is therefore positioned to achieve success instead of failure.

Managing Risk

Successful project teams learn from their environment, adapt to it rapidly, and then predict accurately what is going to occur next. Project teams that can understand and implement actions that minimize uncertainty and maximize stability and predictability can operate equally well in a volatile or stable environment. For most projects, the ability to manage risk is the limiting factor for project success. Preparedness for uncertain events is the goal of risk assessment and management.

Managing risk has two inherently different approaches. The first involves the reactive management of risk; the second, proactive management of risk. MSF advocates a proactive risk management approach. Proactive risk management means the project team has a visible process for managing risks, and the process is measurable and repeatable. With proactive risk management, the team assesses risks continuously and uses this information to make decisions in all phases of the project. The team carries risks forward until it resolves them or it handles them if they emerge as problems.

Maintaining a Fixed-Ship Date Mindset

A fixed-ship date mindset means that a team treats its projected ship date as unchangeable. Essentially the schedule side of the triangle is considered fixed. Once fixed, the team cannot use this side of the triangle for making corrective decisions unless no other option is available.

Adopting a fixed-ship date mindset:

- Forces creativity by requiring the team to implement features in as timely a manner as possible and removing the option of delaying the ship date.
- Prioritizes tasks according to importance. If the team needs to drop features in order to make the ship date, it delivers those most important to the customer.
- Empowers the team by providing an effective decision-making tool. The team makes decisions on the basis of how they will affect the team's ability to deliver on the fixed ship date.
- Provides a motivational goal for the team. A constantly slipping ship date creates morale problems for a team and can ultimately lead to a

developmental "death march," in which team members lose interest in a project because it seems as if the product will never ship.

Peters emphasizes this point in saying:
"There's a thousand different variables that go into shipping a product, the feature sets, the people working on it, how long they're working, a bunch of stuff. All we're trying to do is fix one of them, just one. Of all the thousand variables, let's just fix one variable and let's vary the other 999 variables. When you fix the ship date, you force creativity, you force decisions."
A fixed-ship date is not a forced or dictated ship date that has no bearing in reality. It is a date that the team arrives at through bottom-up estimating and the use of buffer time. This is fundamental to the success of implementing a fixed-ship date mindset because the team must be willing to commit to the date as a realistic and achievable goal.

Breaking Large Projects into Manageable Parts

For large and/or complex projects, the feature set (list of what is to be built) should be broken up into smaller, somewhat independent pieces. Treat these pieces as internal releases or subprojects, each with a defined quality bar for determining when the internal release has been achieved. Think of it as versioned releases within a single project where only the final version is released at the end of the project.
Development teams spend approximately two to four months on an internal release. Each release has time allocated for feature development, optimization, testing, and stabilization. In addition, approximately one-third of the total development time is allocated to the releases as buffer time for unplanned contingencies.
For each internal release, the development team delivers a cluster of features for testing. Assuming that the release is testable, the team goes through a full test/debug/retest cycle, as if it was going to ship the product with just these features. When the code meets or exceeds the quality bar for the internal release, it achieves the zero-defect milestone and the team proceeds to develop the next set of features.
Breaking development into subprojects:

- Allows the team to focus on delivering a smaller and better understood aspect of the project.
- Provides a sense of completion for the team as it achieves each internal release milestone.
- Provides early warning signs of project health because each internal release milestone is an assessment point with its own postmortem review. If internal releases slip, then the team can take corrective action earlier to keep the project on track.
- Increases the overall quality of the product because each internal release has its own quality bar to achieve.
- Allows the team to practice shipping with each internal release so that the actual shipping of the product at the end of the project will be more predictable.

Steve Maguire in *Debugging the Development Process* states:
"It's not the two-month period alone that creates the wins and fosters enthusiasm. It's the thrill of finishing an interesting subproject.
"'Finishing all top-priority items' may be important, but the top priority items don't make up a subproject. They're just a random list of things that happen to be important. There's no motivating theme behind such a list.
"Implementing the charting subsystem is a subproject. All of the tasks that would be involved would relate to that common theme. You might use a task list to remind people of the known charting issues they'd have to handle, but ultimately the theme of the subproject would drive development. The goal wouldn't be for the team to finish 352 unrelated tasks. The goal would be to do everything necessary to fully complete—to 'ship'—the charting subsystem, regardless of whether the tasks it would take were on a list somewhere. The subproject would be in 'ship mode' from the outset."

Performing Daily Builds

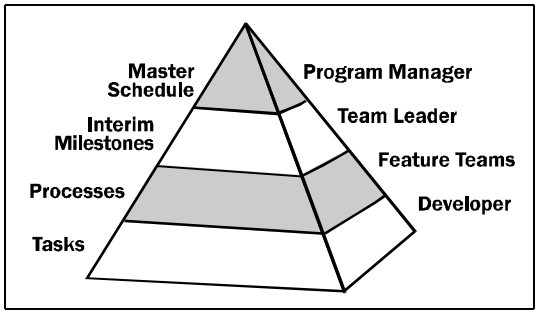
A typical software development team project involves "building" an executable program from up to thousands of different files. At Microsoft, software development teams practice the "daily build and smoke test" process in which they compile every file, combine them into a single executable program, and put it through a "smoke test" to see if it runs. A smoke test exercises the entire system to expose any major problems. The daily build is not valuable unless accompanied by a smoke test. Performing daily builds and smoke tests provides a number of important benefits. This practice minimizes code integration risk by identifying incompatible code early and allowing the team to make debugging or redesign decisions. It supports improved defect diagnosis, making it easier to pinpoint why the product may be broken on any single day. Additionally, it reduces the risk of low quality.
The team must perform the daily build and smoke test each day—not weekly or monthly—in order to produce the greatest benefits. The software being built must work or else the build is viewed as broken and it must be fixed. Performing daily builds and smoke tests is like trying to ship a product every day, which enforces a sense of discipline upon the team.
Standards for daily builds and smoke tests vary from project to project, but at a minimum they should include:

- Compiling all files and components successfully.
- Linking all files and components successfully.
- Finding no "showstopper" bugs that would make the program hazardous to operate or prevent it from launching.
- Passing the smoke test.

Using Bottom-up Estimating

Simply put, "Those who do the work, should estimate the work.." Requiring those individuals who actually perform the work to develop their own work estimates provides two fundamental benefits. The first is that estimates tend to be more accurate because they are based upon experience. A person asked to perform a particular task is expected to have previous experience in executing it or a similar task.

The second benefit is that team members are more accountable for their work. Since they have developed their own estimates, they are more accountable for the success of meeting those estimates. Initial estimates may be high but with milestone reviews, project knowledge and practice estimating, the estimates will begin to reflect the tasks more accurately and will provide the right task level motivation.



Bottom-up Estimating

As the low-level estimates are rolled up into the master project schedule, buffer time is added to ensure that the schedule is attainable. It is this technique that makes the fixed-ship date mind set possible.

Conclusion

Developing software is a complex and dynamic undertaking. As McCarthy observes, "Shipping ordinary software on time is damned hard. Shipping great software in any time frame is extraordinary. Shipping great software on time is the rarest of earthly delights…"

Shipping great software is increasingly difficult because of the burgeoning growth of Internet applications and enterprise-wide multi-tier applications. Successfully managing the development process for these application types requires two important qualities. The first is rigor, which ensures that a repeatable process is followed. The second is flexibility, which allows the process to adapt to a changing environment. The MSF process model provides rigor through milestone-based planning and flexibility from its creative iterative process.

The MSF process model can help deliver projects on time with better results and quality using an engaged project development team.