The Enterprise Application Model

Framework and Language

Version 1.0

September 2026

Tim Mangan

Contents

1. What this is/is not, and why we need it

This is not a product. I'm not building one.

It is an idea. A model that provides a framework that an organization can use to describe and improve upon how they prepare and deliver applications in their environment, and a common language to communicate that.

Large organizations have groups of people responsible for getting the operating system images, desktops, and applications that organizational members use to do their jobs on a daily basis. The names of these groups within the organization, the technologies they use, and the way that they make this happen all vary, but there are purposes, themes, and functions that are common across industries in how they do it.

A common term we hear used in this space is "End User Computing" (EUC), which roughly means everything that connects people to the technology they use to get work done. DEX (Digital Experience) is another popular term right now that extends this. This idea lives as a part of these spaces, no matter what you call it.

This model is an effort that intends to document an organized collection of practices and policies related to how large organizations prepare and deliver applications for use on Windows operating systems. While not attempting to take on the full scope of EUC or DEX (or whatever term Gartner wants to make up next), this work focuses on the preparation and delivery of software components.

This is not a "best practices" document, but one that helps to define portions of the workflows that these large organizations often use.

The goal is to define a technology independent framework that covers a superset of the pipeline that governs the acquisition, deployment, and lifecycle of an application that any given organization will implement for the life of an application.

We also seek to establish a common language for the steps and activities to aid in communicating these processes, not only within the organization, but also for vendors in describing their offerings that might fit into portions of the framework.

We will avoid judgmental terms such as must and shall, which might appear in a recommendation document, but just explain what is possible.

We call this the Enterprise Application Model, a model for how an enterprise might build their application pipeline. And while a fully automated solution might be great someday, the reality is that for most organizations it is still mostly humans clicking on keys and mouse buttons that make things happen today.

With the increased pace of software change we are seeing in organizations, it is imperative that we understand what we are doing today, look to what others are doing to improve what we are doing, and find ways to do it faster. IT needs to redefine itself to empower the organization, not just serve it.

Although we at TMurgent are taking the lead in developing this paper, we seek input from all interest stakeholders. Let's talk!

2. Scope

The image on the cover of this document implies a fantastical automated application pipeline that does not exist. It is meant to be interpreted as inspirational, not real.

This project is an attempt to document all activities that might be taken by an organization starting with the point of the initial request for the application through the retirement and removal of the application from systems.

While specific technologies might be relevant to your needs, this work attempts to be mostly technology independent. Whenever possible, we will describe functions and not the tools.

This documents an adaptable framework, parts of which describe what your organization does today. In some cases, it also introduces a language for communication and common understanding both within your organization and with vendors that want to sell you their wares in this space.

It is understood that most organizations will only use portions of what is described here, and there is no intent to imply that they should.

Organizations today often do not put everything through the policies and procedures that they already have in place. Or may have completely different ways of dealing with different application groups (based on type or organizational effects). In my experience, most organizations today don't really know everything that has been deployed.

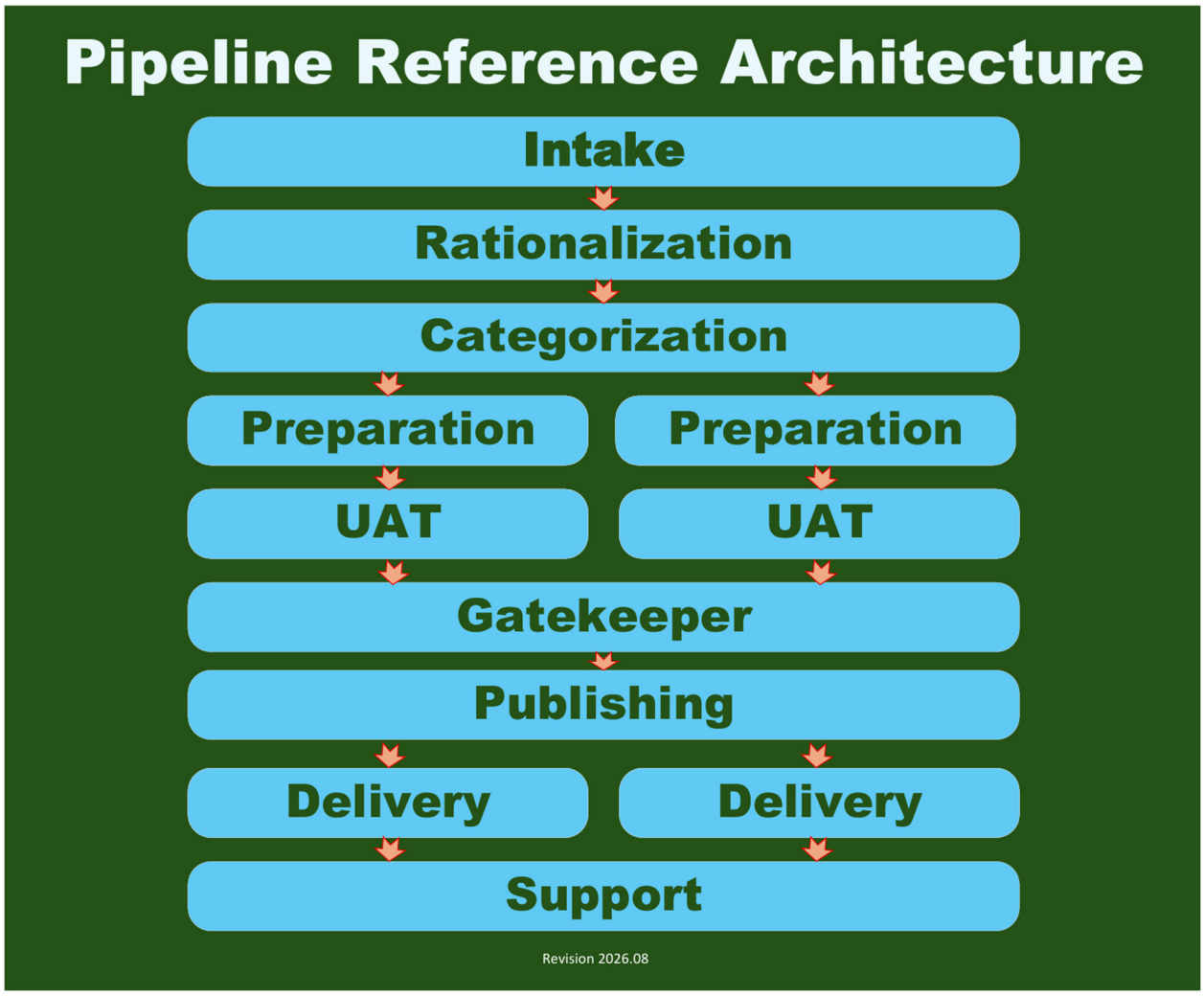
The inaccurate work, duplication, drift, and rogue applications that occur in organizations today lead to reduced productivity and higher support costs for the organization.

Adopting your customized version of this framework, or versions (if need be), and enforcing that all new application deployments follow the model framework will be key to success in increasing productivity and reducing the burden on support systems. There is a "How to Implement" section near the end of this document that is a high-level set of steps to help you adopt and "operationalize" this model.

3. Overview of The Pipeline and Organization of this Paper

We depict the sum of all activities deemed to be within the scope in the form of a pipeline, an ordered set of (potential) activities performed in a sequence that tends to have dependencies on each other. If the word "pipeline" bothers you, just substitute the word "workflow" for it.

This paper will organize the activities described within the major categories of the following diagram. Activities of a given pipeline element might be implemented by different working groups within the organization; the intent is to organize by timeline.



Notice that the pipeline is not necessarily a single flow, but may branch into separate flows where it is necessary. The branch depictions above is quite possibly typical, but not the only way to implement the pipeline.

4. General Pipeline Considerations

Before diving into the various stages, there are some common items to consider that affect the overall pipeline.

4.1 Number of Pipelines

In an ideal world, there is one policy and one pipeline that covers all application requests in the organization.

But in larger organizations, sometimes factors may exist that cause the organization to adopt separate policies and pipelines for different circumstances. The cause may be organizational structure related, technology related, based on tradition, or criticality of the application/change.

4.2 Application Ownership

The concept of an Application Owner is widespread in the field. The definition and implementation of application ownership, however, does vary.

This owner may have some of the following responsibilities:

- Making certain decisions related to this application.
- Subject matter expert.
- UAT tester.
- Initiating or requesting application upgrades, retirement, or replacement.
- Confirming application users and dependencies.
- Maintaining primary business relationship with the application vendor.
- Supporting communication to application users.

In general, each application is assigned with an application owner, although in some organizations there can be a management owner and a technical one associated with each application.

4.3 Enforcement

In addition to adoption of the policies and procedures associated with an organization's customized application pipeline, Enforcement is a key topic to consider.

A well written, communicated, and enforced pipeline brings many benefits such as:

- Everyone knows what they are supposed to do.
- Standards are more consistently adhered to.
- Reduction in drift.

- Well-understood status of request status.
- Better app inventory knowledge.
- Optimized use of licenses for licensed applications.
- Reduction in Shadow IT.
- Reduced duplication of applications with overlapping functionality.

The organization will need to consider when enforcement will be applied, such as:

- All new requests going forward.
- Certain types of requests are exempted.

5. Intake

5.1 Scope

The Intake stage covers all processes related to handling a request for a new application, update or patching of an existing application, and possibly removal of an application, into a trackable item. This includes the period from request expression up to, but not including, evaluation of the request.

5.2 Responsible Group(s)

Personnel from various groups may be involved in Intake (based on the organization).

- *Requester*. This may be an end-user, an end-user organizational group or management, or in some cases from within the IT organization itself.
- *Support*. In some cases, existing support systems, such a ticket tracking systems, might be used to record the initial request.
- *IT Management*. Ultimately, there is likely someone or multiple people responsible for tracking and managing the request to completion. Separate systems from the ticket system might be employed.

5.3 Activities

Requests may arrive in the form of a decision to conduct a major technology migration, or represent an individual request.

Although there may not be a formal process for the acceptance of these requests, often there is. The form of requests accepted within an organization may include some of the following:

- Phone Call
- Email
- IT Support "Ticket"
- Web or tool based forms

Additionally, the organization may be proactive:

- Monitoring vendor websites for new versions.
- Monitoring the CVE lists.
- Use of third-party tools for release/CVE monitoring.

No matter the form or forms of input, it is important to have a clearly documented and enforced intake procedure that will be used for all future application requests. The

organizational policies on intake should ensure that the phone call or email request is properly handled and recorded, just like any other trigger.

The types of information that are sometimes recorded during the intake process include:

- *Requesting Initiator*. Person or group.
- *Application Details :* Vendor, Name, Version, etc.
- Type of Application. The types are usually defined by the organization.
- *Priority* and/or C*riticality*. These may be two different things. The importance of the application, such as a Line-Of-Business (LOB) app, or the importance of the requester (such as the CEO) affect priority. A CVE or down users/systems affect criticality.
- *Requested/Required Date*.
- *Business Impact*. Positive, or negative if this isn't done.
- *Scope/Impact*.
- *Acceptance Criteria*.
- *Business justification for needing the app*.
- *Number of users/devices that the app will be deployed to*.
- *Is it an upgrade request of an existing app, or a new app being introduced in the environment?*
- *Has the app gone through security review within the company?*
- *User names and device names for user-acceptance testing*.
- *Vendor or internal documentation for app configuration/packaging*.
- *Whether the app is required on physical endpoints or is a virtual app (that would be published to end users through Citrix/Horizon etc.)?*
- *Whether it is a licensed app and who manages the licenses?*

The person recording/processing might need to supply some of the items above. This person may also be the one to add additional items into the record:

- *Related ticket numbers*.
- *If licenses need to be acquired*.
- *Designated Application Expert*. May be an end-user or person within IT.
- *Standard Policy Deviation*. An indication that this request forms a deviation from the standard policy.
- *Risk*.
- *Additional items that are organization specific that will be needed downstream*.

A good intake policy will be clear as to what items the initiator of the request is expected to supply.

Generally, the receipt of the request may be:

- *Recorded*. There will often be a tool or place dedicated to this.
- *Reviewed and updated*.
- *Acknowledged with status*. Sometimes requests will be rejected in this stage outright, sometimes in the Rationalization phase.
- *Passed on to the next stage*. When appropriate.

6. Rationalization

6.1 Scope

The Rationalization stage covers the period where investigations are made, and decisions are made about the request. The goal is to filter and prioritize.

A few years back I wrote a separate paper on this topic, which dives into more detail than is covered in this document[1]. In that paper we summarized the goals of typical Application Rationalization strategies to possibly include things such as:

- Software Licensing Cost Management
- Operational cost reduction by removing redundant applications that serve the same purpose
- Operational reduction by identifying troublesome applications, such as those that increase the help desk ticket count.
- Risk reduction by identifying applications or those with a high update count due to software vulnerabilities (CVEs).
- Application Modernization as an aim to improve organization performance.
- Validation to ensure the request reflects a legitimate business need and aligns with organizational norms, rather than being driven by things such as user's personal preference or assumed need.

The outcome of the determination will be a decision to continue with the request, or to reject the request.

6.2 Responsible Group(s)

This will happen somewhere within the IT organization but is highly variable as in most cases it is a small part of someone's main job.

The responsibility may be performed by a single person or shared within many.

6.3 Activities

6.3.1 Filter Determination

The investigation of the request might include the following tasks:

- Determination if the application request is for an application already in use.

[1] Application Rationalization : https://www.tmurgent.com/appv/images/WhitePapers/ApplicationRationalization.pdf

- *Version policies*. How many versions of this app are in production? What do you support (N, N-1, N-2 where N is the latest version from the vendor). What is the out-of-compliance window (time-bound exception policy) for older versions.
- Is the application supported by the vendor, or EOL?
- Discovery of other applications in inventory that may meet the same needs. Prevention of application sprawl will reduce the burden on IT services, and possibly reduce licensing costs.
- *Determination if the application is appropriate for the organization*. This might include:
    - How the vendor handles data (cloud, data sovereignty issues).
    - Vendor reputation.
    - History regarding past CVEs.
- *Cost / Value determination*. There may be a limit, above which a separate approval chain may also need to get involved in this determination.
- *Application dependencies, and impact on other apps and systems.*
- *Has the application been installed and tested with raw files by the SME to ensure it fully fits business requirements and functionality?*
- *Is the application marked compatible by the vendor on the build/version(s) of Windows in use in the environment?*
- *Concerns and policies the organization has over things like data sovereignty and/or AI.*

6.3.2 Prioritization and Severity

Often, organizations will have a backlog of requests and will maintain an orderly queue. Organizations with stronger application policies tend to have two queues, one for standard items and another for the ones that can't wait. Those queues are processed in parallel such that standard items get completed too.

Some organizations will use a single Priority evaluation for placement in the queue. Others will define Priority and Severity as separate values.

Placement within a queue might be based upon things like:

- Is there a known issue that needs to be addressed. Both security and productivity issues might be considered.
- In security issues like CVEs, is it vulnerable (someone said it could be done) or is it known to have been exploited.
- Impact on, or date requirements of, other projects.
- How long before the end-of-life of what is being replaced.

- Is there a reasonable solution available until this request may be delivered.
- *Importance of the requestor*. This is a reality that most organizations have to deal with. But recognizing that the overuse of this as an exception policy leads to increased backlogs of other requests, IT leadership may need to push back on some of these requests. The standards exist to help you reduce the backlog, and abuse of this privilege can have the opposite effect.

7. Categorization

7.1 Scope

The Categorization stage is simply the step in determining the pipelines that will be needed for this application request, and possibly placement in the queues. Recall the parallel sub-pipelines shown in the Overview diagram.

While the goal might be that there is only one pipeline to handle all requests, often an organization will have multiple pipelines to handle different kinds of applications. A given application might need to traverse only one of these sub-pipelines, or multiple pipelines in parallel. In the latter case, consideration in arranging your pipeline might be given around how to prevent work duplication.

7.2 Responsible Group(s)

Often, this may be done by the same group that does the rationalization.

7.3 Activities

Categorization usually is determined based on how the application will be delivered. Examples include:

- Delivery requirements. This may include a separation of applications that require dynamic delivery versus a persistent delivery.
- The need to repackage or wrap the vendor installer to produce a customized solution.
- Packaging format(s) requirements needed for the delivery system, such as an application virtualization or app layering system.
- Delivery Channel(s). This means the tooling that will be used for the delivery. Often this is tied to the packaging formats, but can be independent.
- Whether the app only needs to be made “available” aka “advertised” to end users so they can install as/when needed in future, or does it needs to be forcefully deployed out?
- Does the app need distinct configs (and thus packages/MSTs) per department/location/server?
- Does the app need distinct configs (and thus packages/MSTs) for regular users versus super users or power users of the app? In certain cases, power users need additional shortcuts/functionality related to the app than regular users do.
- Does the app need a deployment-time or runtime flexible configuration script to adapt the application configuration appropriately.

One consideration in setting up policies for categorization is whether an app that will be delivered in different formats and/or delivery channels is a single application record or multiple entries.

The type of application may also play a role in categorization, or in which parts of a sub-pipeline might be needed. Examples include:

- Standard applications. These are well-known applications, possibly already present in the organization. Such applications may be able to bypass security reviews.
- Line-of-Business applications. These can also be considered well known. They often will have quick release cycles with hard limits on the time to deployment due to dependencies of other changes, such as back-end components, that require coordination. These applications tend to have a high and stringent User Acceptance Test defined. When new, or major changes are involved, these applications may require a high level of manual effort.
- Non-standard applications. These applications may need a more detailed security assessment, as well as determination of needed pre-configuration and automated installation information.

When multiple pipelines are needed, in some organizations there will be separate schedules for each line.

A good framework for the pipelines will also allow for handling the following:

- With multiple pipelines in use, it may be possible that some of the steps in these multiple pipelines may be consolidated. If it is possible to deliver all needs with a single packaging effort that might be considered.
- It is also possible that repackaging into a particular format will fail and an alternative path might need a different pipeline.

8. Preparation

8.1 Scope

The preparation phase is where technical work is performed to get the application ready for deployment.

This is the first point in which separate sub-pipelines tend to diverge. In some cases, the same application request may need multiple of these sub-pipelines. Such as an application that is to be delivered to static desktops, possibly through an electronic deliver system like MECM or Intune and also delivered to VDI or Multi-User OSs via something like AppAttach.

When an application will be handled via multiple sub-pipelines, ideally some of the work described in the Activities will be shared. Ideally, the steps to install and configure using the vendor installer and potentially wrapper scripts can be determined once and used in the actual packaging or repackaging of the installer.

8.2 Terminology

Because these terms are used in industry today sometimes vaguely or even mis-used, we find it necessary to define our terminology here.

Preparation is an overall category that includes any and all of the following terms.

Packaging is the act of producing an output that will be used. The technique used to perform the packaging operation may be one of the remaining terms below.

Repackaging is the act of converting the vendor installer into a new installer of some type, either via conversion or package-capture. Repackaging to formats like App-V, MSIX, FlexApp, Numecent, Omnissa Cloud Volumes, and Citrix App Layers are common examples.

Transforms is the term we will use for the creation of a format supported supplemental file that can be consumed by the vendor install file. Examples include MST files for MSI based installers, or .iss or .ini files for certain exe based installer frameworks.

Wrapping is the term we will use for the writing of a script wrapper around the vendor installer. Sometimes this is also called just **Scripting**. PowerShell files are the most common scripting language in use today, sometimes with the help of a PowerShell module built to make this kind of work easier. PSADT and PassiveInstall are well-known modules for this kind of work.

8.3 Responsible Group(s)

Typically, this occurs in a specialized group within the IT organization, often associated with the deployment type. This may be referred to as the "Packaging Group", but when it is called that it means what we call "Preparation". "Desktop Engineering" is another name we hear out there. "Endpoint Management" is yet another term used in some organizations, while in others the term may be used to specifically apply to Thin Client device Management.

8.4 Activities

8.4.1 Understanding Dependencies

Applications often have dependencies for framework or runtime components. These dependencies are typically to be documented as part of the process. This is true whether or not the dependency is packaged with the application. Sources for this information might include:

- Vendor documentation.
- Test installation into a clean OS.
- Test launching the app.

8.4.2 Silent Installation

When deployment includes installation in front of the end-user, determination of silent installation possibilities of the vendor installer will be needed if only wrapping of the installer will be used.

This determination is also needed if automation of a repackaging is needed.

The vendor website will always be the best source for this information.

Often, the vendor may have multiple installer choices. When multiple choices are available, the organization often has a policy to determine order of preference.

- An MSIX based installer from the vendor might be preferred due to the identity, containment, security and maintainability aspects. These installers have an unvarying well-known silent installation as they intentionally do not provide for per application parameters. The format includes a discoverable set of dependencies; however, these may not be a complete set of dependencies so checking with the vendor may be needed. Unfortunately, this type of installer might not be available for the app. And while customization is still achievable via Modification Packages

(used as equivalent of the MST), many organizations still lack the trained personnel that know how to perform that work.

- Many organizations would be looking for an MSI installer. These installers have a common well-known parameter set for silent installations, as well as a per-application set of discoverable parameters from a table in the MSI. While there is the possibility of the MSI installer including dependencies, the most common practice is that the MSI excludes dependencies so checking with the vendor is necessary. Existing skill sets in dealing with MSIs are normally in place in most organizations.
- The exe installer will have necessary parameters that will need to be discovered when the vendor does not document them. As there are at least a hand-full of frameworks available to the vendor to build there exe-based installer, determining the silent and other installation parameters when the vendor doesn't clearly document them can be challenging. Sometimes the exe will support display the syntax by using an argument like /? -? -help or –help. In reality, most of these installers are built using one of a dozen tools to build the installer, and there exist some tools that can analyze an exe to provide the silent installer syntax for this exe. These installers will usually provide all the external requirements, but check with the vendor documentation when available.
- Other forms would include zip files, self-extracting zips, and "portable" apps that have no installer. Portable apps have become a problem in recent years for organizations that want to fully manage the application inventory because typically systems lack the prevention of an end-user installing the app into their profile folder using standard end-user security permissions. This does not prevent IT from using these apps when they manage where and how the apps are installed.

8.4.3 Controlling Vendor Updates

Software vendors like to include automatic updates, or at least automatic update notifications, to their software.

Many organizations need to control what software is updated and when, possibly for regulatory reasons, but also for productivity reasons and for security reasons.

Therefore, the disabling of the update features is often included in this preparation process.

Some organizations will automate the monitoring of vendors sites, and public CVE lists, to determine when new software is available, and to use that as a trigger for the intake process.

8.4.4 Controlling App Features

Software vendors sometimes include features that you do not want to deliver to end users, or a subset of end users in the organization.

Although rarely done, there are some organizations that actively look to reduce the supplied feature set to only that which is required, especially for software delivered to call center personnel.

The latest trend in this space has been to identify and assess/disable AI features/capabilities of an application, although this may prove to be a temporary issue.

8.4.5 Packaging: Re-packaging, Wrapping/Scripting, Transforms

Some organizations will, by policy, re-package the vendor supplied installer into their own custom installer, while others may use a script wrapper around the vendor installer. Either method allows for the customizations around things like:

- Silent unattended installation
- Managing dependencies
- Controlling updates
- Controlling app features
- Eliminating EULA and other early run annoyances
- Adding initial application configuration information for the users.
- Setting audit flags via registry for application record/inventory once deployed.
- Enabling/disabling/configuring startup or first run functionality of the application.
- Ensuring only the required app shortcuts get installed, and at the locations needed (for instance – on the desktop, in start menu, in system tray). This also can include removal of readme and uninstaller shortcuts.
- Updating the application installation directory if/as needed for standardization, and to ensure it doesn't get installed in system folders, or in the user's profile, or at the root on C:

8.4.6 Initial Testing

The software industry has long known that the cost of fixing something that is broken increases as time passes, and especially when more people get involved.

Many organizations break up the testing process of packaged/wrapped software by quickly testing by a single individual or two, before expanding out to the larger population. In some organizations this might be an automated test.

This often means that the person performing the preparation is responsible for some initial testing. This is sometimes referred to as the **Initial Test**, or **Smoke Test**.

This is often faster without involving production delivery infrastructure, which allows for speed and not needing to roll back updates, however many organizations use the delivery infrastructure anyway.

There is a place later in the pipeline for a fuller User Acceptance Test that is usually performed by an application expert using production deployment technology. In my trainings and consulting gigs, I find that I need to remind them that this is part of what the UAT test is for.

The organization may have a policy around how much testing is to be performed. Such activity is often manual in this stage, and keep in mind that the preparer is likely not an expert on the application. They may not, for example, even have access to needed resources like a login/password to an external database to use. However, for updates to known applications where automated testing has already been established for this application as part of the UAT test, this might also be applied here. If not, automation in the form of application agnostic automated launch testers, ones that just search for new shortcuts, launch them and look for error dialogs, might be used.

Especially for well-known applications being frequently updated, especially the in-house LOB apps, a detailed checklist or even automated test might be appropriate in this stage. In some cases, the output (a document/email confirming the tests passed on this version, or even screen-captures still or video images) may be captured and provided to the in-house developer or designated Application Expert. Things like logs might also be captured and part of the reported output.

The organization might also define standards for what states of the outcome of the initial test is. States like “pass”, “fail”, and “conditional pass” are in common use.

8.4.7 Notarization

Notarization refers to the act of someone specifically authorized to verify the identity and validity of the software. This is separate from the functional validations that will follow, it is the organization placing its stamp of approval from a security standpoint.

This is rarely an explicit action of the application pipeline today but implicitly provide the basic function in their processes.

I know I said I'd avoid the word "should", but organizations should consider making it a formalized explicit output for all software preparation.

Notarization work might include looking for digital signatures, or executable hash validation against the installation media or individual components. Documentation of the digital identity of software components (often in the form of XML policy files for AppLocker or Windows Defender Application Control) is a value-added service that App Preparation should be providing to the Security Team.

Notarization implies that there is a practice that identifies software. At a minimum, the notarized software is cataloged and used in tracking activities.

Notarization might also include running anti-malware scans or detonation chamber testing.

The scope of notarization may be at the application package level, but there is also value in extending it to the component level within the package. The use of SBOMs by the vendors can aid this discovery when available.

8.4.8 Documentation

Documentation of the app preparation is also very important. It is highly unlikely that this will be the last time this app will need preparation.

The documentation need not spell out everything, although in the past it was not uncommon to see a document with step-by-step screen shots and instructions for how the app was prepared.

Today, it is more common to assume that the person reading the document will know how to perform the preparation and only wants to see the specifics about this particular application.

Like backups, the documentation requires occasional testing, particularly when new personnel come onto the scene or changes in the preparation practices are made.

Documentation is especially valuable for capturing custom or environment-specific application configurations that may not be obvious, even to the vendor. If that knowledge remains only with the application owner or a single subject matter expert, it creates operational risk.

8.4.9 Uninstall/Rollback

Ensuring an uninstallation and rollback technique for the application is available/created and that it works. Ensuring “clean app uninstallation” could also be a part of this activity – which sometimes needs additional scripting.

There are deployment technologies where this is automatic as part of the tech, or the application does not require anything special, but otherwise this can be part of the preparation process.

9. User Acceptance Testing

9.1 Scope

The User Acceptance Testing, or UAT, stage involves having someone other than the application preparer test out the newly prepared application.

9.2 Responsible Group(s)

This is often an individual person and is someone familiar with the application. This might be a designated end-user of the application, often called the App Expert. This might also be someone within the IT organization as the Subject Matter Expert (SME).

9.3 Activities

UAT typically tests out both the delivery mechanisms and application itself. Ideally, the production deployment methods are used but only assigned to the App Expert.

For some applications, especially frequently updated in-house developed applications that are at the heart of the organization, there may be a written checklist of items to be tested. For other applications, the App Expert "wings it", testing the items they feel are most important to complete their normal duties and/or things that have been a problem with the app in the past.

It is important that final UAT is performed in a production environment, and not in Dev/Test/Staging, because especially in case of client-server apps, the app parameters/configs/functionality etc. may be different in low-level environments compared to when it is in production.

The use of automated UAT is also growing. Automated UAT is not generally good enough by itself today for many organizations, and they tend to use it as a pre-filter to human in-person testing if it passes.

Not every application is likely to require a UAT. Establishing a policy to help personnel identify which applications need a UAT is wise. Such a policy may target:

- Key Line of Business (LOB) applications.
- Applications with complex multi-app workflows and integrations.
- Applications subject to Regulation. For example a customer that performs manufacturing for drugs is often highly regulated.
- Heavily used apps that update frequently (browsers, plugins, etc).

10. Gatekeeper

10.1 Scope

The Gatekeeper stage is one that is likely to exist in organizations today, even if it is more informal in smaller organizations.

This stage acts as a review and check that all work has to date has been performed to the standard.

10.2 Responsible Group(s)

In some organizations, there may be a formal group of people that perform this review, often called a "**Control Review Board**" that approves any changes in the production environment, including the application to be delivered. This not only serves as a review that organization standards around what has been outlined in the prior stages of this document have been met, but possibly that a risk assessment of the change has been performed and may consider the needs to control the order and flow of changes being made with an eye towards the security and/or stability of production.

When not formed by a group, this may be a sign-off by the person responsible. In the future this is an area that automation, especially in the era of AI, may be sufficient to perform for most applications once the organization implements their version of this model.

10.3 Activities

Control Review Boards typically hold regularly scheduled meetings where the App Prep team can apply for approval.

11. Publishing

11.1 Scope

The Publishing stage is when deployment information is identified and recorded. This generally consists of the WHO (who will get the application) and HOW (how it is delivered).

11.2 Responsible Group(s)

Most often this is a function performed by the team responsible for the deployment tooling and infrastructure, and the organization today might consider this stage and the next stage to be a single stage.

In some organizations, that group that controls the delivery infrastructure will only determine the HOW. In that case a different group is responsible for the WHO. This might be line management or the group also responsible for application licensing.

11.3 Activities

The assignment of applications to end-users may be done in different ways.

- Define an “application object” in the identity directory. Then assign users, groups of users, or devices, to the application object. Define the application object also in the deployment system and point it to the object in the identity directory.
- Define an application object in the deployment system and assign usage in that tooling.
- Consider and apply special settings in the delivery system, such as dependences, task sequences, and supersedence (aka “does something else need to be removed first).

The collection of installation assets and adding them to the deployment tooling is also part of the activity.

Often, there are some documentation activities to be recorded.

12. Deployment/Delivery

12.1 Scope

The Delivery stage is when the approved application gets deployed.

12.2 Responsible Group(s)

The team responsible for the delivery infrastructure.

12.3 Activities

Some organizations use delivery methods that support a partial deployment initially.

- This may involve an initial deployment to a select small subset of users.
- This may involve a more complicated ring-based deployment that can often include multiple expanding rings.
- This may just be work-group based, so that only one group is at risk from a bad deployment.
- This may also involve coordinating the deployment with other groups in the organization, such as the server team (in case of client-server apps) to ensure a coordinated release occurs.

When a partial release deployment of any of these forms are used, once you are satisfied with feedback from the first group/ring, you can expand to the full deployment.

The deployment group is typically responsible also for monitoring the deployment and second line of defense (after the help desk) for issues reported to the help desk until the deployment is complete and deemed stable.

13. Support

13.1 Scope

The Support stage starts upon Deployment, yet remains active through the eventual retirement of the application.

13.2 Responsible Group(s)

The “Help Desk” is the primary. In larger organizations there may be multiple escalation levels within this organization.

The Deployment Team also has responsibilities, when asked for.

The App Prep Team may also be called upon.

As is the Application Expert.

And there may be a group responsible looking at application usage data.

13.3 Activities

Once deployed, some of the following activities are typically employed.

- Monitoring/Enforcement
    - Usage/Misusage
    - Licensing
    - Last Time Used
    - Persistent tickets
- Allowed List Enforcement
- Support
    - Triage issues
    - Solve if can
    - Escalate if needed.

14. Summary

This work is only useful if your organization adopts it.

What that means is your organization taking their current policies and organizing them in the terms outlined here. As part of that you would likely examine what you do today and make minor tweaks initially as part of implementing your version of this model, while making a plan to extend that over time to include items you find compelling.

Embracing automation and tooling as part of the adoption, or down the road, will be necessary to keep up with the growing application demands.

IT has an opportunity to recast the current efforts in this space, not as a necessary cost center, but as a group that is empowering the organization and its workers to achieve more.

For example, adding help to the security team by delivering to them the software identity of all applications as part of the preparation process so that the organization can move to locked down systems where only approved software can run is becoming essential. Training users not to click on things only goes so far, baked-in prevention is needed.

IT leadership may need to lean on those ideas to secure the resources that will be needed to achieve their goals.

15. How To Implement

This Application Model is a superset of what an organization does today. Adopting the model involves the following steps.

1. Review current policies and procedures, as well as organization structure. As part of a project to implement, consider if this review might include all the stakeholders early on.
2. Identify where those policies and procedures fit into the model defined in this paper to build your copy of the model that reflects your organization today. This creates your initial documentation.
3. Review your current model with all stakeholders to ensure accuracy, making sure that reviewers stick to reality and not what they would like it to be. Minor changes can be considered, but for bigger changes collect the feedback and emphasis that there will be another phase of the project for those items.
4. Get signoff and by-in from all parties on the published version and implement.
5. Review sections of the model from this paper that are not in your production model, as well as other big changes identified but deferred earlier. Identify sections that are of interest to improve your operations. Work with stakeholders to prioritize the items and define how to implement the changes by adding them to a proposed updated model. After buy in, then publishing and implementing the new model.
6. Set up a regular review that repeats step 4. Be sure to look to see if there is an updated version of the paper as part of this step.

16. Author Notes and Acknowledgements

Tim Mangan runs TMurgent Technologies, LLP. TMurgent has a rich tradition of providing research papers to the IT community in the application preparation and delivery space, but also for training, software, consulting, and even packaging services to Enterprise organizations around the world that are using App-V and MSIX technologies for application delivery.

Although my name might be on the cover, this is an effort of community work taken from my interactions with, pestering of questions at, and puzzled amazement at IT Pros at so many organizations that manage to do this work, sometimes with only a vague formalized idea of how they really get it done.

Thank you to all who endured my early presentations on the concept of this work, while I was looking for input.

Special thanks go to some people who have substantially contributed to this paper, both in feedback and ideas:

Bob Kelly (Juriba)

Sucheta Gawade (Quorum Health)

The many customers who have shared their practices with me over the years.