AmberReef Blogs

Application Packaging – Managing The Critical Path

Contents

Introduction

I hear a lot about application packaging factories. These might be off-site capabilities provided by a 3rd party partner or even an internal function labelled as a factory. I would like to talk about this capability less as a physical entity with inputs (source media) and outputs (application package) but more about the internals, the factory based practices and processes that will actually add the value to application virtualisation packaging. I believe that application virtualisation is a different thing to MSI packaging. The former creates an execution environment the later delivers a payload. They are as different as virtualised and physical server infrastructures. Whilst they achieve the same end result (Users running applications on EUC (end user compute) devices), applying a common set of processes and procedures to their creation will only devalue the capabilities of both. Once you have mentally decoupled the two and consider the processes etc. that can be achieved without the restrictions of the pair, then you might be surprised how much more effective the delivery of applications to users can be,

What is a Factory?

The benefits of a factory are that it produces a "product" faster than it can be produced by an individual crafts man. A master carpenter can produce 1 "Perfect" cabinet in the same time that a factory might produce 100 "Perfectly acceptable" fit for purpose cabinets.

The most important consideration in the above paragraph is the notion of "Purpose". The purpose needs to be defined apart from the people that are going to deliver the output. If you put 10 master carpenters in a room and told them to make cabinets, human nature and professional pride would probably lead to zero output. This is because skilled people tend towards the perfect. There would be lots of meetings about design and methods, ultimately resulting in either a set of compromises that no-one is happy with and the output is likely to be less than the sum of the parts. If you set the same group a number of parameters as a purpose, such as cost, durability, size and a timeframe, then these same people will more likely deliver something great.

Tolerances

There are tolerances in the manufacturing processes that allow for deviations from "Perfect", and there are processes for ensuring that the tolerance levels are maintained affectively. This is an ongoing process where continual improvements are made based on knowledge gained. Things are unlikely to be "Just Right" the first time but will, over a number of iterations, move towards perfect.

Throughput Management

It is also important that an item on the production line, which is outside of the tolerance levels, does not interfere with the output of the production line. In other words – The production line does not stop to make repairs or alterations to an item that does not meet the standards. These are rerouted to other paths in the factory.

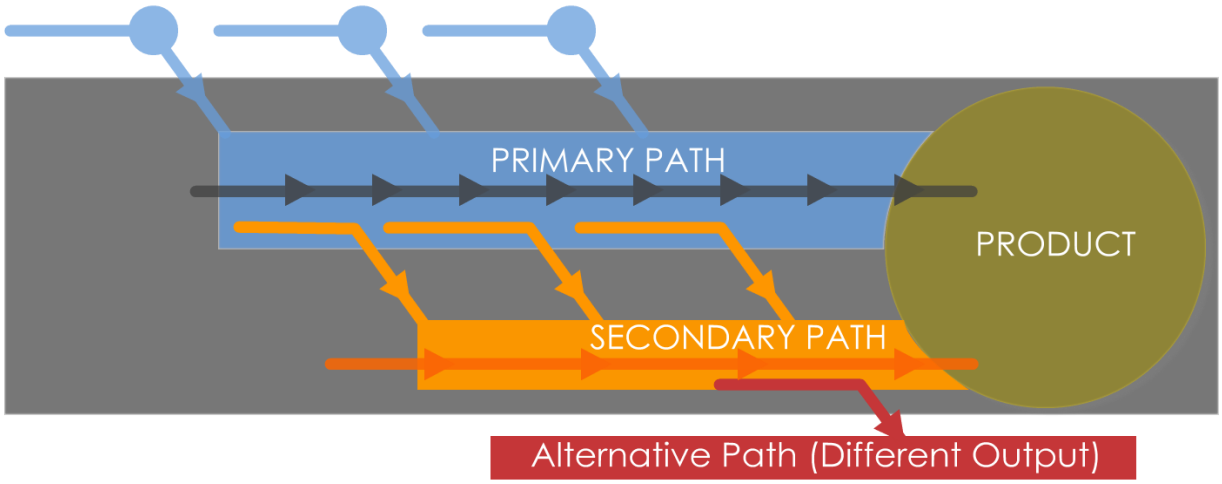
Buckets

The production line should flow continually. Components should be produced at a marginally greater rate than the production line consumes them. This allows the process that creates the components to be offline for maintenance etc. whilst the production line consumes the "Stock pile".

A bucket must never be empty nor full.

Demand

The factory must be able to manage its throughput based on demand. It must be able to scale up production and affectively scale it down during peaks and troughs (Christmas for example), without adding more complexity or massively increasing overheads.



The diagram above is a simplified view of these principles where the "Primary Path" runs without disruption, the "Secondary Path" runs at a slower rate but still has a continual output flow whilst failed products or by-products are dealt with outside by their own set of processes.

Applying these principles

It is no surprise that the new generation of project management methodologies (Lean, agile, DevOps) have their roots in Manufacturing. I believe that learning from industries that have been around longer than ours is an important step. There is little point in learning the hard way. I think this article (http://bit.ly/1gNEYMz) captures that pretty well

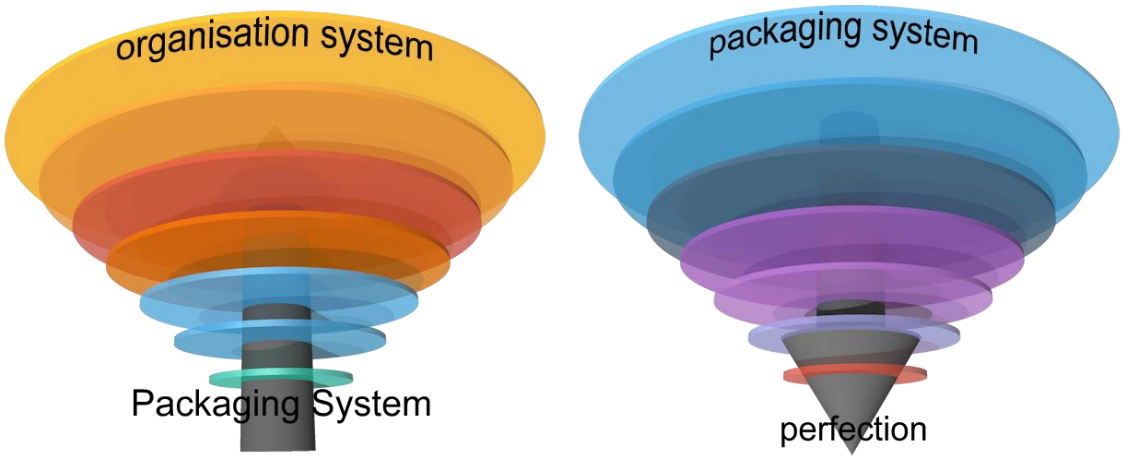
Setting the "Purpose":

"…focus on the way that a system's constituent parts interrelate and how systems work over time and within the context of larger systems." (Ref: http://bit.ly/IzuZ1C).

If you dissect an organisation into system parts, and we accept that systems exist within other systems, and rarely if ever as a closed single system, we can build a simple view like such:

"The Organisation exists within the "Market section" it trades in. The IT function lives within the Organisation system. The Desktop System lives within the IT System and the Packaging System lives within the Desktop System etc."

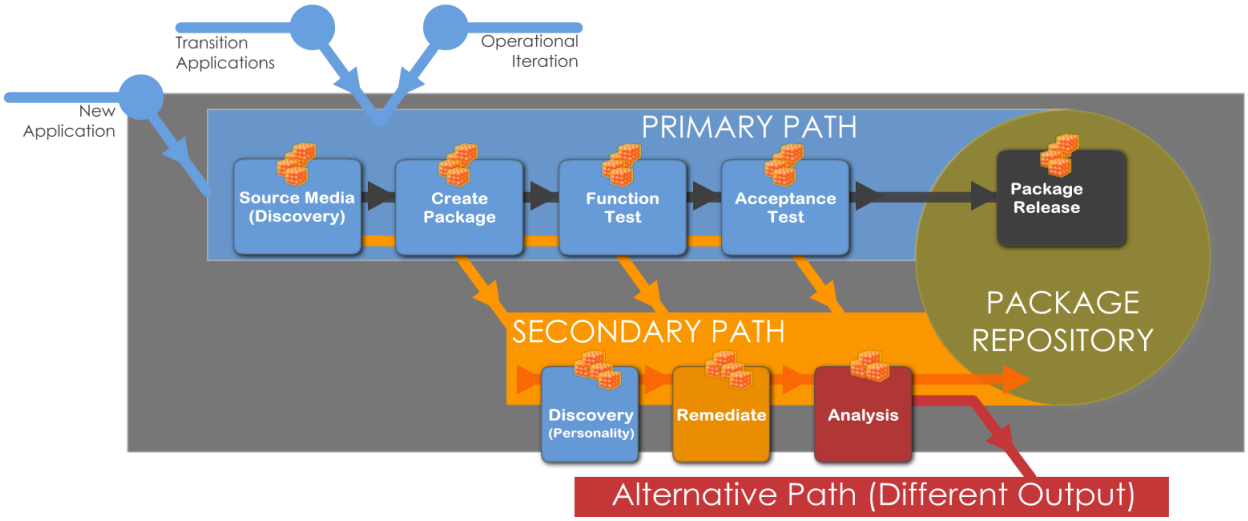
If you define the Packaging Systems "Purpose" in line with one of the parent systems that it resides in, then the focus of the Packaging systems will be upwards and the processes etc. can be defined in relation to the bigger picture. If you define the "Purpose" at the Packaging system level then the natural tendency is to focus inwards "Towards Perfection". This will produce processes and procedures that are not in tune with the higher level systems nor any sibling level systems. If the system that creates nuts and bolts in a factory is designed to create the perfect bolt then there is a pretty good chance it will also end up creating these too slowly and the reason for creating the bolts in the first place, the Factories Product, will be impeded.



There is a place for the "Spiral to Perfection", however this must always be kept in check, and in line, with the consuming and parent systems. It is the adoption of iterative, continual improvement processes that ensure this balance is achieved.

So how can we apply some of these considerations to our virtual application packaging processes?

It is worth discussing the context of "Hard", "Medium", "Easy" applications a little. An Application should only be considered "Hard" once. It will tend to be considered hard because there is a lack of knowledge about how the application is installed and configured. Once the package has been created the first time and that "Knowledge" is captured, the application is easy. In context of the image below, I would like to change that definition into applications that are "KNOWN" and "UNKNOWN". Only New Applications into the business / Packaging Process should be considered as "UNKNOWN" and therefore "Hard". All other applications will have a degree of knowledge associated with them (it might be hidden somewhere in a department member's desk draw and hard to find). Other than "NEW" applications all applications in the packaging process can be split between the remaining two entry points, "Operational" and "Transition" and described by degrees of effort.



Operational applications in the Packaging process are applications that are currently running within the business, have been through the process before and are having an application update applied. The degree of effort will largely be determined by the incremental change in the application (a 0.1 increment in version is likely to be less effort than a 0.5 increment). There should be an incentive within the business to run a frequent, continual improvement program on all client applications, and remove the disruptive big version leaps that are often found today. It is during these smaller incremental iterations that the "TOLERANCE" levels increase and the resultant output improves both within the packaging process and the user's experience (the "Spiral to Perfection").

Transition applications are those that currently exist and run in the business. They are entering the packaging process because of an external influence. Examples of Transition influences are:

1. Operating system major update
2. Business driven platform change (departments moving to RDS from Laptops)
3. A hardware refresh to x64 architecture and associated operating systems
4. A change in package formats (MSI – App-V, App-V 4.x – App-V 5.x etc.)
5. A combination of some / all of the above.

Transition is the current focus on most peoples' minds. But if the processes and methodologies put into place now solely focus on this factor then there will no possibility to add ongoing benefits to the business of any new technologies introduced. Businesses are looking for more flexibility in the User Workspace arenas. They are looking to allow users to work in many different ways and key to these capabilities are the "Client" applications and how users can interact with them. There is a whole other article to follow about "Workspace Services" but as a foundation it is key to understand that the application is more important than the platform that it is running on, indeed it is the application's capabilities that will determine the Platform Services that a user might be able to consume.

So back to the Topic:

Tolerances

Use repeatable steps to create packages. This includes cleaning a package. Learn what are the common steps taken when cleaning a package and make them repeatable, build on the knowledge from the manual processes. In the initial instance do not expect to make the Package "Perfect" – just "Perfectly Acceptable". Use the App-V Template (sprt / appvt) files to exclude file and registry locations that are known to not break the package. Use this for every package and learn what else might be applicable, then implement these changes in a continual improvement process. Do not react if the settings break some packages, leave these to the

Secondary Path. Create some rudimentary scripts that can be run against each package that will clear out the Installer directory. These scripts do not need to be overly intelligent in the 1st instance, they just need to ensure the package size is not excessive (see here).

These tolerances are acceptable for application virtualisation packaging because you are creating a runtime, an execution environment, not a delivery payload. If an application does not virtualise and follows the "Alternative Path" then there are different tolerances that require you to strive for a nearer "Perfect" package.

Throughput Management

Do not get hung up on the Packages that do not "Just Work". Pass these to the "Secondary Path". Create a set of processes that allow a light touch in the "Critical Path". Concentrate on getting the applications that "Just Work" or require a minimal amount of remediation through the "Primary Path". This approach will also have the effect of reducing what looks like a big job (particularly when working within the "Transition" context), into something that is more manageable.

Having a continual flow of packages released WILL positively impact the programs that are consuming your output. At this point in time this is likely the "Desktop Refresh" program. This program will be able to make more informed decisions on the users to migrate because there will be a constantly growing list of ready applications.

I have been involved in far too many projects, that have an end goal that involves changing the users execution environment, that have stalled due to a lack of available applications.

Buckets

There should never be a time when one of the sets of resources are waiting for work. Daily visibility of the "Buckets" is crucial. If the packaging engineers are waiting for work then your processes should allow for temporary re-allocation of some of the resources to other buckets. It is always better for the throughput to slow down than come to a stop.

Do not wait until you have every application discovered before you start the production line. Start with what you have, and feed the packaging bucket as applications are found. The discovery information does not have to be fully complete. It is better to have 10 incompletely discovered applications in the bucket than no applications at all. If the application is missing crucial information, that is what the "Secondary Path" is for. Use the skills of the engineers in the "Secondary Path" to discover what is missing and feed this information back to the main discovery process. This will help the enable a more targeted Discovery Process.

Remember – Creating a package that works but is later found to not be needed is a much much better use of time than not packaging it and finding out it is needed, because you will then be introducing a much more visible and detrimental delay. You are much more likely to have packaged an application that is needed, but an older version. This is still valuable as you will have converted a "NEW" application into an "OPERATIONL" package and that uplift in iteration is far less disruptive.

By ensuring that there is always a continual flow leaving the "Packaging Process" you are in affect "Filling" one of the Buckets for the consuming service. (Systems in Systems). You will also be largely decoupling the Packaging Service from the disruptive waterfall demands of any migration program. These normally come in the form of "We are moving marketing. Where are their applications?" This level if interaction between the two systems will result in the complete destruction of the "THROUGHPUT". Filling the migration bucket by proactive factory processes will mostly enable the reversal of this relationship.

Demand

Most demand is currently based on a "TRANSITIONAL" program. I feel that the previous topics largely cover the considerations around being at the top of the Demand curve. One often neglected consideration when in the middle of a transition program is that BAU still exists. "OPERATIONAL" applications will still be entering the process. These will need to be addressed in both the current delivery model and the "Future State" delivery model. The longer the Transition is running the greater the negative affect this dual requirement will have. I have seen situations where attempting to manage this element alone can bring a transition program to a standstill. There are ways of minimising this affect, sometimes it might take some "out-of-the-box" thinking and some political wrangling. Mostly knowing it is an issue, and remembering half way through the transition that its effect is about to start escalating will save a lot of tension and delays. However the only way to truly minimise the effect is to shorten the Transition timeline.

The difficulty that will arise within the packaging processes is, once the hard work of migrating all users in the transition is done, there will be a large sigh of relief, and exhale of "Phew – That's done", the current "Transition" team will be disbanded and all of the new technologies and processes will be handed over to BAU.

Due to the nature of Program / Project driven transitions, no matter how well intentioned or fastidious the program is regarding documentation and knowledge transfer, the introduction of technologies is focused on delivering the objectives of the project. Once the program team is disbanded there will be a loss of knowledge. Very rarely will the program have devised processes / procedures or operating models that fit operational requirements without a degree of change. The state of the application processes should, after the current phases of "Big Bang Transitions", be in a position to greatly reduce any similar Transitions in the future. The operational model of the packaging factory / processes should be based on continual improvements and future state readiness.

Summary

One of the best ways for an organisation to blend skills and knowledge into BAU from the Transition Program is to have the operational team start the future state investigation of all currently released application versions. This is beneficial to understanding applications that may not work on future state systems (think windows 8.1 has now been released) allowing remediation steps to begin early, but also allows the operational teams an "Out of Line" opportunity to bring their skill sets up to speed. The net result of both of these benefits is that when the organisation is ready to introduce the next "disruptive" platform change, the Application Service will already be in a position of knowledge that will enable a far smoother migration. Indeed – An application service that is in a position of knowledge might even enable Desktop Platform services to adopt a continual improvement process, thus removing any need for disruptive "Transformation" programs in the future.

To achieve this there is a real need for automation and factory processes to be introduced into application packaging. Otherwise the resources required to manage both the "OPERATIONAL" application iterations and the "FUTURE STATE" knowledge accumulation will be equal to the combined resources of the transition program and BAU.