A Pattern Language for Release and Deployment Management

RICK RODIN, Pace University, rick.rodin@verizon.net
JON LEET, Pace University, jonathan.leet@gmail.com
MARIA AZUA, Pace University, mariaazua@yahoo.com
DWIGHT BYGRAVE, Pace University, dbygrave@gmail.com

A release is a collection of authorized software changes that include new functionality, changed functionality, or both, that are introduced into the production environment. Agile software development results in many small releases delivered as needed, as opposed to big-bang releases that deploy large amounts of functionality at regularly determined intervals. Agile software development without a detailed standard process model encompassing the overall release management activity can result in an emergent release schedule (which is almost as bad as not having one), potential bottlenecks for resources, and problems handling genuine emergency releases. Changes to software systems must occur in a disciplined and controlled way. Otherwise lack of control over the delivered changes will lead to deteriorated system quality. Change Management is often in conflict with Agile in that Change Management requires a big, heavy, schedule, dates, content, and deployment resources specified well in advance. This is a major attraction of big-bang releases; that their deployments seem to be easier to manage. In a fast changing Agile environment the survival of the software organization may be affected. There is no reason to abandon Agile development for the sake of a disciplined and controlled release process. Our proposed pattern language captures the expertise necessary to create an orderly, well thought out, end-to-end release management process that should help deliver software releases in a disciplined and controlled way within an Agile development environment.

Categories and Subject Descriptors: None

General Terms: Software Maintenance

Additional Key Words and Phrases: Release Management, Patterns, Compliance, Software Risk, Intellectual Property

ACM Reference Format:

Rodin, R., Leet, J., Azua, M. and Bygrave, D. 2011. A Pattern Language for Release and Deployment Management.18th Conference on Pattern Languages of Programs (PLoP), Portland, Oregon, USA (October 2011), 13 pages.

1. INTRODUCTION

The following patterns describe best practices for managing software deployment within an agile enterprise. These patterns focus on the core release management activities and the intent is to develop these patterns further. The relationship among them is depicted on the following diagram. A flow exists, starting from OBTAIN CERTIFICATE OF ORIGINALITY ensuring legal compliance, continuing to implement RELEASE SCHEDULER, followed by RELEASE ACCEPTANCE. After the release has been accepted sometimes it requires APPLICATION RETIREMENT or an EMERGENCY RELEASE to fix a problem on the field. DEPLOYMENT PREPARATION occurs next. EMERGENCY RELEASE has its own form of RELEASE ACCEPTANCE.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission. A preliminary version of this paper was presented in a writers' workshop at the 18th Conference on Pattern Languages of Programs (PLoP). PLoP'11, October 21-24, Portland, Oregon, USA. Copyright 2011 is held by the author(s). ACM 978-1-4503-1283-7

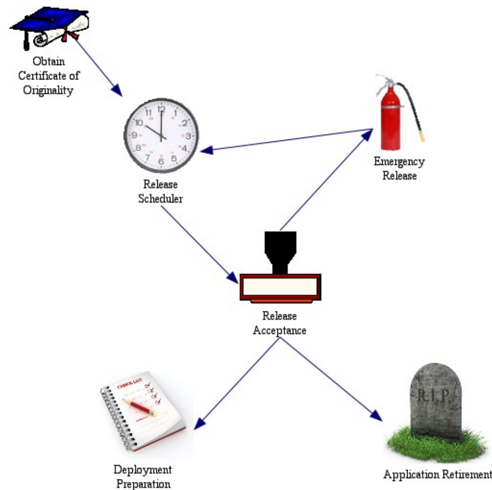


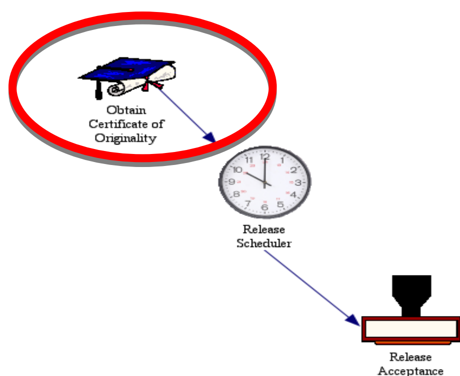
Figure 1: Pattern Relationship

2. THE PATTERN COLLECTION

This paper contains the following patterns:

OBTAIN CERTIFICATE OF ORIGINALITY: Proves that the software is original and bears no license infringement.
RELEASE SCHEDULER: Determine the release timeframe for rapidly changing, high volume of releases.
RELEASE ACCEPTANCE: Guarantees release's regulatory compliance.
EMERGENCY RELEASE: Handles unscheduled, urgent releases.
DEPLOYMENT PREPARATION: Cover the all the steps for a well structured deployment.
APPLICATION RETIREMENT: Removal of software from the production environment.

2.1 OBTAIN CERTIFICATE OF ORIGINALITY



Context: A company releases original software on a regular basis. Throughout all phases of the development process, proof is required in order to establish that the software is original and that no license infringement has occurred. Additionally, new security compliance laws require these entities to track code provenance. Open Source software is typically forbidden when intellectual property is closely managed. A company's responsibilities include awareness of these legal requests and regulatory compliance. Final

legal approval is only the culmination of the various steps and stages in this process. Software companies need to manage their intellectual property in this manner; however any company that creates its own software should consider using certificates of originality.

Problem: A company requires proof that software is original and meets regulatory restraints, before it can be used in any production capacity or sold.

Forces: Due to compressed schedules many programmers forgo to document code provenance or rely on control tools that don't record code contributions with sufficient information or granularity. In addition every day we get more security regulatory compliance laws and rules related to code tracking provenance. The combination of stringent release cycles, limited resources, and ever-increasing compliance regulations create a perfect storm that is forcing businesses to follow a certificate of originality best practice that encourage education.

Solution: A company's legal department or other office governing intellectual property rights will supply a form to fill out and sign. Such forms are of various lengths, requiring one to assert in writing that they have not copied their work from anywhere else, violated existing copyrights, and that it is original work created by the submitter. Often a certification number is assigned once the form is accepted by this office, and this number can be used as part of the signature to sign the binaries. This is not an exhaustive list of elements to testify to, such a form may require one to answer many questions testifying to the originality of their work. All require the signature of the submitter. Such a signed form constitutes the basic certificate of originality and it will be maintained by the company's department governing intellectual property. Check with your company's legal department to see where such forms are located and the proper rules for submission.

Tools exist to validate that no Open Source software has been used. Find one that works for you and run it.

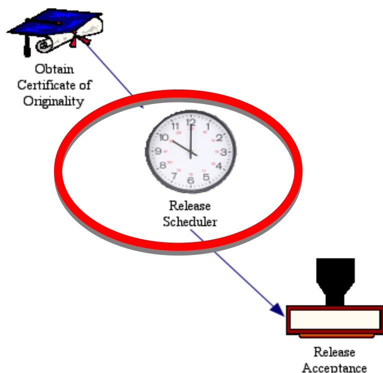
Following the completion of the certificate of originality form and open source validation, further activities include the education of programmers, managers, or any information technology practitioner on regulatory compliance requirements for tracking code provenance and the steps to obtain a certificate of originality. The company must strive to be compliant to legislation governing copyrights and intellectual property. New employees will receive training in compliance as a part of their orientation. Post compliance requirements on an easy to access place to facilitate members of the team to become familiar with the requirements. Once these individuals complete their training their work will be monitored. Audits conducted by the legal department will check samples of work to see that they meet the originality criteria.

Additionally, steps must be made to document the development process, in an effort to establish due diligence in the prevention of infringement on intellectual property. It is important to review code during development, so early identification of infringement can be made and avoided. This will eliminate wasted resource utilization lost to unusable code, in addition to providing a concise and detailed report to facilitate the process of achieving final legal approval during a compliance review. Once everyone has been educated on the code provenance requirements, and they are performing their due diligence, the RELEASE ACCEPTANCE pattern can be applied to guarantee that the other regulatory requirements are being adhered to.

Resulting Context: Code is released that meets the legal requirements necessary to deploy in production or to be sold, meaning that it is original and satisfies regulatory requirements relevant to proof of originality. If items are found that fail to meet originality requirements then the items will be withdrawn and administrative action will be taken against the individual responsible. More comprehensive audits will then be required. Some releases that fail to demonstrate originality may still go undetected, and this is an unresolved force. After completing the various duties and responsibilities required during the development process and gaining final legal approval, it is important to follow the RELEASE SCHEDULER pattern. Other related patterns important to follow for compliance are RELEASE ACCEPTANCE, EMERGENCY RELEASE, and DEPLOYMENT PREPARATION.

Rationale: Failing to obtain a certificate of originality for a software solution could represent deployment delays in addition of risking compliance failure and costly penalties. Getting declined for a certificate of originality request or failing a code tracking provenance audit can be expensive. The result of applying this pattern will be a well-educated workforce on code tracking provenance requirements and as a by-product you will save money and avoid deployment delays.

2.2 RELEASE SCHEDULER



Context: After moving past OBTAIN CERTIFICATE OF ORIGINALITY it is determined that multiple updates to an application will be required to maintain the operational status of that application.

Problem: In an environment where changes happen frequently releases can become chaotic, threatening to overwhelm deployment resources and causing deliverable dates to become uncertain.

Forces: Before further activities can take place such activities must be identified and sequenced with associated duration(s). If activities are missed the performance progression could be affected. Other possible forces could be as follows:

- Loss of network resources which results in financial loses
- Unavailable, services which could cause the business to not function
- Different groups wanting changes all at the same time
- Activities done concurrently that should not be done in or around the same time
- Disorder and inefficiency reduce the success of the deployment

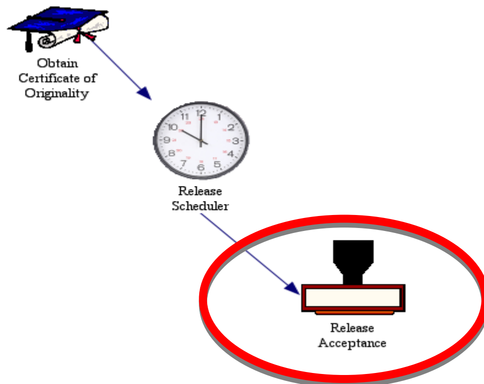
Solution: Once the determination is made that the system being built or the system in
place will support more than a onetime business process then a process needs to be put in place that can see all the upcoming releases in the pipeline, determine or verify their deliverable date, and then engage deployment service providers and other support resources. Service providers and software developers will estimate how long each release will take and in what sequence to deploy each module. An overall timeframe may thus be built for all releases. A detailed schedule may then be printed and distributed. This will allow everyone to know what is happening when, and will reduce the likelihood of resources, physical and human, being unavailable or in conflict, log jams or other conflicts, and deployment sequence problems. It probably will not eliminate all these issues but it will reduce them to a manageable level.

Resulting Context: Releases will take place in a more orderly and efficient manner. Concurrent activity will be easier to plan for and should take place in the right order. Activities that must occur in isolation, or be tested before another activity can take place, or simply not occur when something else is going on, will be easier to manager accordingly. Things will not happen perfectly in sequence, but the planning that goes into the use of this pattern will make a great improvement. It becomes highly unlikely that really

catastrophic results from releasing items out of sequence will occur. Emergency Releases will not fit well into this pattern either and have their own scheduling issues, and hence an entire pattern is devoted to these.

Rationale: The development and introduction of a RELEASE SCHEDULER reduce if not eliminate the possibility of a schedule conflicts and eventual work and performance degradation. Having a RELEASE SCHEDULER significantly contributes to a smoother release process and eventually release acceptance.

2.3 RELEASE ACCEPTANCE



Context: You work for an organization regulated by Sarbanes Oxley, The Federal Reserve, general legislation governing copyrights, and others. Such entities are requiring that software cannot be released to production without proof of testing, business approval, intellectual property ownership, and a back out plan. Part of your job is to guarantee regulatory compliance coordinated with the RELEASE SCHEDULER pattern, that is, before releases are scheduled.

Problem: Releases are not being reviewed prior to deployment and such unchecked activity could lead to the failure of internal audits, or the issue of audit warnings. Furthermore this brings on the risk of being fined for deploying software without approval, test evidence, certificates of originality, and back out plans. Non-compliant software release could jeopardize your standing with your regulatory agencies is in jeopardy.

Forces: It is extremely important for regulated companies to follow their internal and external compliance rules. Failure to comply with these regulations may result in fines, legal action, or a diminished corporate rating. A firm's information technology environment is one of many regulated areas. These regulatory rules frequently include requirements such as having a designated approver attach a formal statement saying they accept the release, accept its risk, and accept its delivered functionality. Other documentation such as backout plans and test evidence is often required to be included with such approvals.

The full range of intellectual property ownership often needs to be addressed, usually in the form of certificates of originality. Failure to include such documentation, or having it incomplete, is costly in terms of time spent, actual monetary fines, legal actions, and overall company rating. To further complicate matters, these regulations are updated frequently and communicated poorly.

It is not uncommon for software developers to be unaware of the latest rule changes, or may work with other developers assuming that the other attached the documentation when no one did. Sometimes people simply make human errors and forget to attach the documentation. Others are sloppy in their work. The deployment team receives the request and misses the documentation for the same reasons, and then deploys improperly documented modules. When auditors review all the deployments that occurred over the

past year, and find releases without the proper approvals or other documentation, they report it as a compliance failure.

A well implemented automated workflow that does not permit unapproved or otherwise document deficient releases to pass to an approved release state can also serve. While helpful, these workflows are too rigid to stand on their own. If approvals are questionable, but acceptable, and documents are acceptably late as in cases of Emergency Releases, or others, such automated workflows tend to hinder the process rather than help.

Solution: Create a role in the organization for a person to act as a Gatekeeper and have this person use workflow tracking software that includes stop gates requiring checks for the presence of these approvals and rejects releases lacking appropriate documentation. The gatekeeper will apply the APPROVER VALIDATION[1] pattern, and may also need to apply the EMERGENCY RELEASE pattern. Additionally, the gatekeeper must validate the certificate of originality. The gatekeeper will activate an "accepted" state in the workflow once the required documents are present. No release may be deployed to production without being in an accepted state in this workflow tool. In its gate keeping activity the Gatekeeper may also reject releases. Rejected releases will go back to the software project manager with clear instructions for resolution.

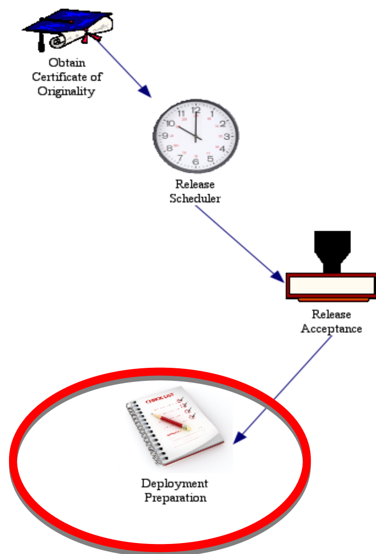
Resulting Context: When an auditor reviews the closed deployment records the releases will meet audit requirements. Exceptions may exist due to human error on the part of the gatekeeper. Some releases might completely circumvent this process and such undocumented releases will present another problem.
Many, even most, people requesting a release will completely follow the regulations most of the time. They take this part of their work seriously and you can largely depend on them. You can probably trust the quality of their work as much as you trust a designated gatekeeper. An extra check provided by this pattern is still valuable and warranted, and the good side is that you only have to check such quality work once. You will check the lesser quality work at least twice. Conscientious people will appreciate the double checking, and they will also appreciate the reputation they will gain for being so thorough. Still it only requires a few people making mistakes to cause the problem addressed by this pattern

Rationale: Regulatory compliance is important and compliance failure is costly. Provision of test evidence, backout plans, and user approvals also add some qualitatively good discipline and tracking. Failed audits are expensive and using this pattern will save you money. The end result will be well tested, authorized software on the production environment.

Occasionally a gatekeeper will make mistakes and let through unapproved releases. It is this individual's job to be very thorough and can be managed to that end. It is more efficient to manage a single gatekeeper than to try to rein in an entire organization. The gatekeeper becomes the one who has to answer to the auditors and his boss about each release and this motivates quality. The gatekeeper specializes in the regulations while the software developers do not, and this specialization leads to greater effectiveness.

[1] The Approver Validation pattern has not yet been written. The core idea is that one must confirm that the appropriate approver has signed off on the release. Approvers vary depending on who the release is for, the contents of the release - particularly if it contains sensitive data, when it will be released - especially if it must be released during code freeze periods, and many other factors. A separate pattern guiding the approver validation activity is therefore warranted.

2.4 DEPLOYMENT PREPARATION



Context: Before any software build is released in an environment, it must be prepared and accepted. RELEASE ACCEPTANCE plays an intricate part of the decision to deploy software. The Acceptance of the release must occur previous to the deployment preparation, which legitimizes the software build and makes it appropriate for deployment. This makes the preparation the next logical step in the release process. After the deployment preparation is done, the release that follows is normally seamless.

Problem: Software packages are reaching the deployment state with components in the wrong format, developed with inadequate security, without resources to deploy them, without necessary documentation, and without adequate testing.

Forces: The working environment is made up of several different functional groups with unique responsibilities. Omitting the proper communication and due diligence before a release will most likely cause the releaser (The person conducting the release) to infringe on the one of the functional group area of responsibility and in most cases this will result in pushback from the group being infringed on. One must make a determination on how it will be released in its operating environment, whether it would be securely or insecurely, binary or generic format, from an internal or external source, via the network or via a disc, amongst a variety of other factors. The software build can then be formatted to the release or customer specifications; copied to disc, FTP/TFTP server, moved to the released subnet/server on the network, if required, compressed, etc. Some of the other issues that could be experienced range from the loaded code not being able to run because it is not in the correct data format. However, once the correct preparation is done, testing must also occur to ensure the proper preparation of the deployment.

Solution:
It is necessary to follow several steps to prepare a software release for deployment. By following these steps, a company can minimize potential damage that could occur during this process. To prepare a release the following must be considered. These items apply generally and consist of the major items to validate prior to the module’s deployment:

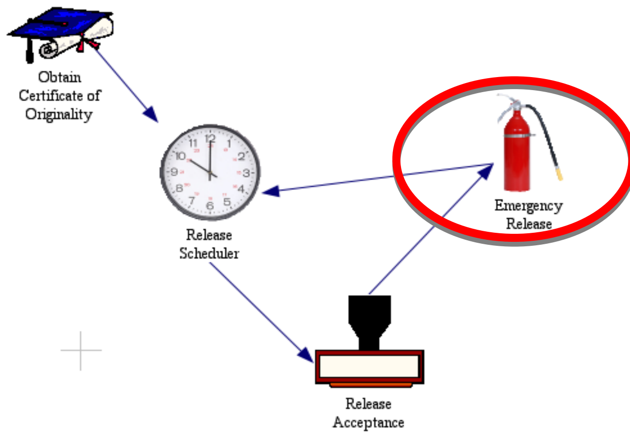
- Security concerns
  - Network Access during deployment
  - Release Server access with directory rights
  - Ensure correct file/data checksum
  - Key generation/access before attempting to deploy
- Data format
  - File Type

    - Compressed or not for delivery
- Media Access
- Ensure availability of support personnel during and after deployment
- Pre-deployment testing
- Appropriate accompanying documentation in the case of government releases.
- Check the version of the software
- Test the deployment of the module in a production-like environment to insure it will run in the actual target environment

Resulting Context: Confusion about release group areas of responsibility will be minimized. A module will now exist that is appropriate for its operating environment and is in the right format. A deployment test will have provided confidence that the module will run in its target environment. This module may now be released securely or insecurely as warranted.

Rationale: Once the proper Deployment Preparation is done, releases normally run smoothly from an activity process flow perspective, and in most cases technical problems get uncovered as well. The time spent preparing for a deployment is always time well spent, and the more time is spent preparing the smoother the deployment.

2.5 EMERGENCY RELEASE



Context: You have an unscheduled release that must be deployed to production immediately and your release resources are committed to other tasks. No one has reviewed how this release will interfere with other planned releases, and it is unknown if testing has been adequate. This immediate release is required to fix, and carries great urgency. Due to the deployment teams being committed to other projects and thus constrained, the actual timing of the deployment is unknown. While development wants the release deployed as soon as they are ready, the deployment team may be physically unavailable prolonging the outage, degraded service, or security breach. This pattern is used for fixing problematic software only, not for new functionality that the business simply wants faster than the usual workflow permits.

Problem: You need to create and release software within an unusually short timeframe to resolve a service outage, degraded service, or security breach. This pattern is limited to releases required for these reasons only.

Forces: A developer, customer, or customer facing part of the organization seeking to deploy an emergency release will immediately encounter some formidable roadblocks. The release will not be on anyone's schedule; therefore they are unlikely to have the time in their schedule to release it and cannot be

assumed to be prepared to handle it. Furthermore, it is unlikely that the release has been tested adequately and may therefore lack the correct form of test evidence.

The approvers designated by audit compliance may not be immediately available to approve the release. This means that the release is effectively out of compliance; that RELEASE ACCEPTANCE might block the release for lack of correct test evidence, delayed application of OBTAIN CERTIFICATE OF ORIGINALITY pattern, and overall approval. The release must proceed even if the officially designated approver is not immediately available to provide signoff. Compliance with Sarbanes Oxley, the Federal Reserve, or other regulatory agencies is always necessary, however in the case of disruptive problems or security breaches an accommodation with the real needs of the organization must be made. All the regulatory paperwork may be completed in a timely manner after the fact. The regulators usually understand this.

Due to the module's unknown relationship to other planned releases, and rushed testing, its effects on the production environment could be negative, leading to the further disintegration of the production environment.

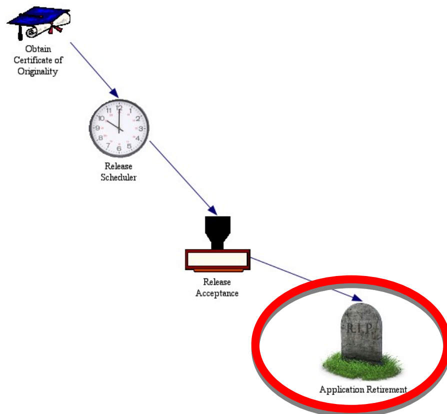
It is possible that the testing will be adequate and the deployment team and all the approvers will be available. Such a situation will actually accelerate the release to production, making this a positive force. It still will not be on the weekly or other regular schedules, meaning that the deployment teams will be unprepared for it. Emergency releases are often small and simple, having a minimal impact on the deployment team's resources.

The risks of deploying an emergency release must be weighed against not deploying it. An outage, degraded service, or security breach is usually costly and generally not acceptable. Delaying the repair of such a system due to lack of release acceptance or scheduling problems is likely to result in further damage. Any action that improves this situation is desirable and implicitly approved. It is still important that all releases adhere to compliance rules.

Solution: A process to support the bypass of the standard gates, scheduling tasks, and the temporary waiving of approval requirements should be created and managed by Release Management. Contact Release Management to engage this emergency process. Release Management, together with development, will assess the deployment requirements and make the arrangements with the deployment resources so that they will be prepared to elevate the release when required. Doing so will engender a process for emergency releases. Higher authority may be requested to reschedule planned releases to accommodate this emergency. Release management will have a list of the day's releases and can arrange a review of this module's relationship to other scheduled deployments. It may not be possible to contact the designated approver in a timely manner, but the release should be able to proceed under a form of "implied consent," with the approval obtained later. OBTAIN CERTIFICATE OF ORIGINALITY pattern may also need to be applied later.

Resulting Context: A rapid restoration of software services with the ability to accommodate emergency deployments without degrading system quality or entropy. Some deployment resources will be disgruntled. If possible, a project retrospective should be held to reduce future occurrences of similar emergencies. All the forces may not be resolved. Any constraints on the time of release resources will still exist and may be made worse by this. A long testing period is impossible and testing may not satisfy all normal quality requirements. How the release will interact with existing software may not be clear until after it is released, thus opening the potential for further releases. The release might not actually fix the problem and another emergency might be needed. Solutions will vary depending on the organization. It is possible to cycle through this pattern many times.
Rationale: Software quality is not always 100 percent perfect and it will become necessary from time to time to deploy software to fix unexpected service outages, degradations, and security breaches. Such fixes will be unexpected and can be disruptive. To minimize these disruptions and maintain the integrity of the existing environment an effective emergency release procedure must be followed.

2.6 APPLICATION RETIREMENT



Context: As the life-cycle of an application approaches this end, the importance, significance, demand and effectiveness diminishes. This becomes apparent, as the computing requirements for the ever changing business environment force the need to retire various different pieces of software. Additionally, more sophisticated and modern software suites may replace or eliminate the need for a specific piece of software with aging capability or limited functionality.

Problem: At times, an application will need to be removed from a production environment. How can an organization remove a piece of software from a production setting, without impacting the overall functionality of an application, software suite, or system?

Forces: It is the nature of software releases to become obsolete or lacking appropriate security after a particular duration. At times, the need to retire an application will present itself. This is usually based on a newer, more appropriate solution or the expiration of need. Whether the application will be replaced with an additional solution, or the application is no longer necessary, proper procedure must be employed at the time of application retirement. The tradeoff in the retirement of an application is the possibility that during the retirement, critical failure will occur. This will significantly impact overall functionality, and it must be understood that even in the best cases, the application may not be restored to its original state.

Solution: There are various items, which in most situations must occur before the actual removal of the retired application. These instructions are incredibly important to follow in a step-by-step fashion. This will ensure that an interruption to critical processes will not occur. By following the steps below, a piece of software can be removed properly and safely, without experiencing critical downtime caused by unforeseen issues.

- Identify all users
    - Notify all users
    - Identify dependent services
- Create and implement alternative for dependent services
    - Identify proper application removal procedures
    - Develop and execute tests to ensure minimal services interruption
- Archive data
    - Identify requirements for data archiving
    - Extract data from existing system
    - Store data and make accessible through pre-determined methods
    - Import data into replacement systems
- Release final communication to user community before application removal
- Remove application
- Test to ensure minimal interruption is maintained

In the event that users are affected by the software retirement, special consideration must be made. This could include a review of the software retirement, a possible discontinuation or removal, or discovery of secondary solutions, if the software is necessary to daily operations.

Furthermore, if dependent services are identified, additional consideration on discontinuing the retirement must be made. All services and downstream application must be analyzed to prevent the incidental disruption of service. If this is the case, a decision must be made whether the software retirement can be continued successfully. In some cases, the only alternative may be to leave the software in place, and continue investigating other means of replacement.

It may occur that identifying all users or systems, which will be impacted, may not be possible and alternatives must be produced. In these cases, special attention must be made to the post-implementation phases, during which resources must be in place to respond to a notification of negative impact from an unknown entity or entities, downstream from the application being removed. Without anticipating potential unexpected service interruptions, the application retirement team may be unprepared for a critical outage, and as a result, may be incapable of effectively correcting the issue in a reasonable time frame.

A MONITOR COMMITMENTS pattern may be necessary to provide a detailed process for monitoring user and system failures resulting from the application retirement. In any event, a proper roll-back procedure must be in place to ensure that services can be restored, if alternatives cannot be implemented in an acceptable manner. This may possibly be the result of insufficient alternative solutions, an unmanageable duration for a corrective action, or the inability to determine the necessary requirements for a complete solution.

Archiving the data is important for various practical and legal reasons. All data necessary for future operations, or data that is required to be archived for a particular duration, must be identified. Following this exhaustive search and analysis, the data must be extracted from the existing system, stored, and imported into the solution systems that will require access to this legacy data. In most case, best practices would suggest that we keep a consistent copy of the data in all situations, as it is often hard to identify what data needs to be archived.

Finally, if testing of all related services fail, a process must be set in place to replace the application with another one or quickly reinstall the removed application. This situation should be addressed and planned thoroughly with the needs and functionality of the software in mind.

Resulting Context: There may be a need for an EMERGENCY RELEASE at this time of the retired software or a further developed version of the new application or service. If critical services are lost, an emergency deployment will be required to correct the issues with the overall functionality and system operation.

Rationale: This design is based on experience recorded with previous application retirements. It is the unforeseen users and services that will experience an interruption in the normal daily operations. With proper preparation, most of these issues should have been identified previous to the retirement of the application. The better the ability to prepare for the application retirement, the greater chance the rollback will occur without interrupting critical functionality of peripheral applications or services.

3. CONCLUSION

The purpose of this paper is to present a core set of patterns serving as best practices for release and deployment management. It is a step in the creation of a larger, more comprehensive pattern language assisting release managers operating in agile environments. In such a setting it is important for release management to be agile too, with the ability to plug in best practices as needed. This pattern language provides this capacity.

4. ACKNOWLEDGEMENTS

We would like to thank our shepherds Steve Berczuk and Lise Hvatum for their very insightful feedback, comments, and support...

…And Dr. Joe Bergin for teaching us everything about patterns while making the process challenging and enjoyable.

5. ABOUT THE AUTHORS

Rick Rodin
For the past 5 years Rick Rodin has been a Release Manager at CIT, a mid-sized financial firm. CIT grew by acquiring businesses, each with its own development shop, which CIT then had to integrate onto a shared infrastructure and regulatory environment. The actual result was a lot of chaos that lead to severe and persistent outages of important business applications. In 2005, Rick created a Release Management service that stabilized this highly variegated, silo behavior, and otherwise chaotic development and release environment. Rick has deep skills in the area of Release Management and software development best practices as well as great experience as a change agent. Rick is regarded as a release management expert with passion for IT transformation. Rick was also a software developer for JP Morgan a few years ago and understands release and deployment management from a developer's perspective, which is a valuable experience as a Release Manager. He has also done a lot of software support and appreciates release management from this perspective as well.

Jonathan Leet
Electronic Commerce Implementation Specialist at General Electric Information Systems for a short period of time and then moved to a technology/engineering role at a small firm specializing in the design of machine parts used in robotic manufacturing. After a considerable duration, Jonathan headed west to work as an IT Coordinator for a national mortgage brokerage. Close to four years later, the company collapsed under the weight of failed sub-prime leading and he returned east to re-entered the manufacturing arena as an Information Analyst.. Jonathan is currently an Analyst at InterComponentWare, Inc. (ICW), a software company in the medical data connectivity space. He finds his role challenging and has assumed the role of IT Coordinator for all North American executive operations.

Maria Azua
Maria Azua has worked for IBM for the last 21 years and is currently the VP of Advanced Cloud Solutions and Innovation. She is responsible for cloud technology development, solutions implementation, deployment and operations of high profile customer POCs. Also she managers the operations of the IBM Cloud Innovation Lab and she chairs the Cloud Innovation Council which manages IBM cross division cloud technology efforts. Prior to this she was VP of Cloud Enablement for the IBM Enterprise Initiatives organization. In this role she was responsible for the development and deployment of the Common Cloud Platform Living Lab share services, as well as the IBM Smart Business Development and Test on the IBM Cloud. She also was responsible of creating a vibrant and engaged community of ISVs, Business Partners, and technical community that fosters IBM cloud computing IT methodologies and applications. Maria is also the author of The Social Factor – Innovate, Ignite, and Win Through Mass Collaboration and Social Networking (2009 IBM Press, Pearson Education), a book about social networking based on Maria's pioneering leadership in social networking tools at IBM. Her contributions to social networking tools, B2B solutions and IT transformation are widely recognized throughout IBM and the wider IT community.

Dwight Bygrave
Program Manager with over 21yrs experience on IT Infrastructure design, deployment, move and support. This would be infrastructure as it applies to Data Center Construction, HVAC, Power, Cabling, Servers, Router/Switches, and various Operating Systems. He started out as an engineer in the Banking/Financial Market (Citibank, Lehman Brothers, Banker Trust/Deutsche Bank, Merrill Lynch, JP Morgan) and moved to the carriers (AT&T and MCI) then Pharmaceutical, and now he is working in the Federal space doing

Program Management on large government programs. His Data Center knowledge and experience goes back about 17 years when he began maintaining and building these facilities. He also has extensive experience managing teams and complex operations developing and deploying telecom infrastructure projects with the Federal Aviation Administration (FAA), Verizon and several others. He manages projects with budgets in excess of $150Mil. Dwight holds a BS (Physics), BS (Construction Management) and MBA. He is PMP, ITIL and Six Sigma (Green Belt) certified.