Everything You Need to Know to Master Release Management

By Kate Eby (https://www.smartsheet.com/content-center/author/Kate%20Eby) | June 13, 2018 *(updated March 16, 2023)*

Today, software engineering is a fast cycle of developing, testing, deploying, and supporting new versions of software for increasingly complex platforms.

While most organizations are not updating that frequently, coordinating the development and release of software versions represents an ever more challenging task.

That's why release management is a growing discipline. In this guide, you'll learn everything you need to know about release management, including the latest trends and expert tips. You'll also find out how to handle release and deployment management with the IT Infrastructure Library (ITIL)/IT service management (ITSM), Agile, continuous delivery, automation, and other approaches. Free templates and checklists round out the resources.

What Is a Release in Software Engineering?

In software engineering, a release is a new or modified software and the process of its creation. A release constitutes a fully functional version of the software, and it is the climax of the software development and engineering processes. Alpha and beta versions of the software typically precede its release.

While alpha and beta version launches may also be called alpha or beta releases, in the singular form, release generally refers to the final version of software. You may also see releases referred to as launches or increments.

Most organizations identify releases with a unique set of numbers or letters that update sequentially. This naming process is called *software versioning*. There are no industry-wide rules for how these unique identifiers change from release to release, but each company consistently follows its own internal standard.

What Is the Release Management Process?

Organizations improve the quality, speed, and efficiency of building or updating software by focusing on release management. This is the process of planning, scheduling, and managing a software build through the stages of developing, testing, deploying, and supporting the release. Techniques like Agile development, continuous delivery, DevOps, and release automation have helped optimize release management. The velocity of this process has accelerated recently, to the point where several years ago Amazon passed the mark of 50 million code deployments a year (https://www.allthingsdistributed.com/2014/11/apollo-amazon-deployment-engine.html) — more than one per second.

Release management is a relatively new discipline in software engineering, but it's also growing rapidly thanks to swift innovations in technology. As a discipline, it draws from both traditional, business-focused project management and technical knowledge of the systems development life cycle (SDLC) and the IT Infrastructure Library, a set of practices for IT service management.

The Objectives and Benefits of Release Management

Done effectively, release management increases the number of successful releases by an organization and reduces quality problems. Productivity, communication, and coordination are improved, and the organization can deliver software faster while decreasing risk.

These improvements mean the team can repeatedly produce quality software with shorter times to market, which allows the company to be more responsive to the operating environment.

Release management also helps standardize and streamline the development and operations process. The team implements auditable release controls, thus creating a repository for all releases throughout the life cycle. Having a single, well-documented process that must be followed for all releases increases organizational maturity. Increased

standardization and the focus on product allow teams to draw more useful lessons from experience and apply them in future releases.

Operations departments appreciate the increased coordination with developers because there are fewer surprises. They can now avoid feeling that a release has simply been "thrown over the wall" from development, leaving operations to fight fires and "patch and pray" because of short deadlines. There's also more of an opportunity to resolve configuration issues between the development and operating environments.

In short, release management breaks down team barriers across multiple functions in an IT organization. As a result, you can improve product delivery holistically.

Gabriel Gutierrez (https://www.linkedin.com/in/gabriel-gutierrez-abb2a914), a Solutions Architect at Boeing and former Release Manager at Costco, says that when executing a release, it's important to give adequate consideration to the impacts on other areas of the organization. "Companies must provide a common forum where changes are fully vetted and put through thorough architectural and design reviews that eventually lead to integrated testing. Cross-functional and technical reviews are essential to minimizing the inevitable pain after go-live. A smooth post-go-live operation is a critical success factor of any release," he explains.

A Brief History of Release Management

The rise of release management comes from software engineering's shift from project-based to product-based offerings. Under the project-based development paradigm, software developers would view each release as a project, not a product; fully developed software largely signaled the end of the developers' role.

Over time, however, the software development process came to more closely resemble the product cycle in which products are supported, improved, and repeatedly relaunched over a long lifetime. In this framework, the release was not the end goal of development, but rather a transition point for support and revision.

With this increased complexity came a greater need to coordinate the phases. That's why current release management practices are inspired, in part, by the principles of business-focused project management, which extends to after-sales support and further development.

Release management is part of the larger IT discipline of change management (/essential-guide-everything-change-management), which deals with the inherent turbulence of software development, where stakeholder requirements seem endlessly fluid and the landscape of compliance and regulation continues to shift. To navigate this environment, software engineering processes must be well synchronized, and release management helps achieve this. (Note: Change management is not to be confused with organizational change, which is the reshaping of culture, turnover of people, and internal restructuring of organizations.)

A "common pitfall is disregarding the importance of a well-thought-out change management process," says Jen Dunbeck (https://www.linkedin.com/in/jendunbeck), a release manager at automated IT services provider BitTitan. "A change control board's or change advisory board's significance cannot be underestimated in a release. A key function of their job is to help the organization assess risk and impact with impartiality. They also help root out technical dependencies that may not have been evident as individual contributions to the deployment," she adds.

To develop your change management process, you will find it helpful to have standard ways for proposing project changes and recording changes as they are approved and made. Jumpstart the effort by downloading the following free templates for a change proposal and change management log.

Change Proposal Template

CHANGE PROPOSAL TEMPLATE

PROJECT NAME		DATE CREATED	
PROJECT MGR.		VERSION DATE	
ORGANIZATION		VERSION NO.	0.0.0

CASE FOR CHANGE		
PROPOSED CHANGE	Detailed overview of proposed change	
WHY CHANGE IS REQUIRED	Detailed overview of reasons necessitating the change and how the change corresponds with the organization mission	
INTENDED OUTCOME	Detailed overview of resulting achievements and benefits	
ESTIMATED TIMEFRAMES	Anticipated timeframe for prep, plan, consult, implement, and eval	
ADDITIONAL FACTORS	Consider any other factors crucial to the successful implementation of proposed change: need for change awareness, work environment climate, previous changes, etc.	
ESTIMATED COSTS	Complete table, below	
STAKEHOLDERS IMPACT	Identify stakeholders and the potential benefits and adverse effects for each	
	POTENTIAL BENEFITS	POTENTIAL ADVERSE EFFECTS
STAKEHOLDER 1		
STAKEHOLDER 2		
STAKEHOLDER 3		
STAFF & OPERATIONS IMPACT	Identify areas likely to be impacted by the change and the potential benefits and adverse effects for each	
	POTENTIAL BENEFITS	POTENTIAL ADVERSE EFFECTS
PROCESS		
TECHNOLOGY		
STRUCTURE		
OTHER		

APPROVAL			
PARTY PROPOSING CHANGE		SIGNATURE	
CONSULTING PARTY		SIGNATURE	
ENDORSED BY		SIGNATURE	

ADDITIONAL COMMENTS
Include any additional comments

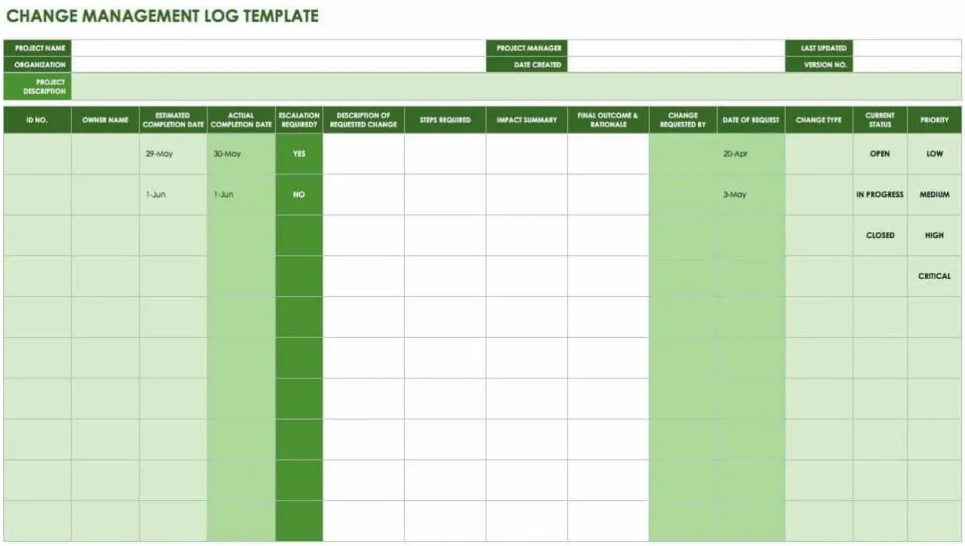
COST / BENEFIT ESTIMATE			
ESTIMATED PROJECT COSTS			
RESOURCE	DESCRIPTION	PROJECTED EFFECTIVE DATE	ESTIMATED COST
Staffing			
Consultation			
Assets			
Technology			
		TOTAL	$ -

STRUCTURAL CHANGES				
ELEMENT	DESCRIPTION	PROJECTED EFFECTIVE DATE	ESTIMATED COST (+)	ESTIMATED SAVING (-)
New Hires				
Redundant Positions				
Promotions				
Demotions				
		TOTAL	$ -	$ -

POTENTIAL NEW HIRES

A change proposal outlines the type and scale of change, and is often the first step in a change management process. Describe why the change is needed, expected outcomes and impacts, time and resources required, and any other factors that need to be reviewed. This template also includes space for adding descriptive information as well as sections for calculating costs and benefits.

Change Management Log



CHANGE MANAGEMENT LOG TEMPLATE

PROJECT NAME						PROJECT MANAGER					LAST UPDATED		
ORGANIZATION						DATE CREATED					VERSION NO.		
PROJECT DESCRIPTION													
ID NO.	OWNER NAME	ESTIMATED COMPLETION DATE	ACTUAL COMPLETION DATE	ESCALATION REQUIRED?	DESCRIPTION OF REQUESTED CHANGE	STEPS REQUIRED	IMPACT SUMMARY	FINAL OUTCOME & RATIONALE	CHANGE REQUESTED BY	DATE OF REQUEST	CHANGE TYPE	CURRENT STATUS	PRIORITY
		29-May	30-May	YES						20-Apr		OPEN	LOW
		1-Jun	1-Jun	NO						3-May		IN PROGRESS	MEDIUM
												CLOSED	HIGH
													CRITICAL

A change management log is a document that tracks who requested what change and when, the status of the change request, its priority, and resolution information. If you need a more thorough record, include other details such as the type and impact of the change. This log template is designed for keeping track of vital information so that change requests can be easily prioritized, addressed, and referenced later.

How Release Management Works: An Overview

Not all software is created equal. Depending on the program's functionality and application, the software engineering process varies in rigor and intensity.

For example, software that's designed for use in aircraft undergoes rigorous testing. You will have to create extensive documentation and implement multiple fixes if you don't meet narrow performance tolerances. Functional requirements and quality metrics are stringent, and the software has to go through many layers of approval before it ever gets near a plane.

A disciplined release management process will help ensure that software is built, tested, and delivered in line with the main stakeholder's stipulations. The team will check the software to see whether it does what it's supposed to do and whether it's ready on schedule.

Release management combines the following elements in one central repository:

- The business imperatives of giving customers what they need when they need it
- The technical tasks of quality assurance and compliance
- The management of software artifacts

From the repository, the artifacts are released to a client production environment for application deployment. So, you can think of release management as the glue that holds together the interconnected but distinct tasks that comprise software engineering: design, development, testing, deployment, maintenance, and business and customer requirements.

A release comprises more than just the core software engineering functions. While developers obviously view a release as the successful delivery of a finished product, there are several supporting business activities that are also involved, such as training salespeople and customer support staff and advertising and marketing the new release. These activities need to occur in sync with the release tempo, adding complexity to the release management process.

Moreover, release management is not just for newly designed software. Modified software goes through the same process to ensure that the stakeholders get what they want.

The actual release can be done manually, or, increasingly, it can be automated, depending on how mature the release management process is. (Amazon uses an automated release process to achieve a new release every second. Historically, weekly, monthly, and quarterly updates have been more typical for software development teams.)

Releases can come in many different forms: as a physical product in a box, as a download from a website or server, as a push to a device from a mobile app store, or as a seamless update to a web-based application.

All software engineering groups — whether big or small and across all industries — use release management to smooth the process. They may apply it to software used internally or sold as a product. Software engineering groups big and small employ release management across all industries, and they can use the resulting software to run their own product or systems, or to sell as a product in itself to clients.

One of the development approaches most conducive to release management is called *DevOps (/devops)*, a term formed by combining development and operations. As its name suggests, DevOps attempts to increase coordination between developers and operations in software engineering by using a combination of automation

and monitoring at all stages of software development to decrease time-to-market and improve customer satisfaction.

Key Terms in Release Management

To master release management, you'll need to understand commonly used terms.

Development Work Order: This is a work order for the development or modification of a software application or system.

DevOps Team: Since the whole point of DevOps is to increase coordination between the development and operations functions, creating a separate team makes the term a misnomer. Despite the name, the term usually refers to key team members on both the development and operations sides who work to coordinate the two functions.

Installation Work Order: Similar to a development work order, an installation work order is for the installation of a software application, system, or infrastructure component.

Product Owner: The product owner on a development project is the main stakeholder. They usually represent the business or the product's (eventual) users, and they define the vision for a product.

Project Manager: A project manager takes charge of the direction for a single product. They're responsible for defining the product roadmap and vision and for negotiating deliverables. Typically, a project manager's core responsibilities do not extend beyond a single product or closely related family of products.

Release: A release consists of one or more release units.

Release Manager: The role of the release manager is to plan, coordinate, manage, and schedule all the items that comprise a release. All these items do not necessarily have to originate from the same product. For example, this manager must also coordinate work on any other products that are designed to integrate with the new release.

Release Policy: This is a set of rules for how to deploy releases to the live operational environment. Different release policies apply to different releases, depending on such factors as impact and urgency.

Release Record: A release record documents the history of a release, from the planning to the closure of the development process.

Release Unit: This term refers to a set of configuration items that a team simultaneously tests and releases into the live environment to implement approved changes. A *configuration item*, in turn, is a component of an infrastructure that's under *configuration management*, which is the process of making sure that a product's attributes and performance are consistent with its design, requirements, and operational information.

Service Owner: A service owner takes on high-level accountability for a specific IT service. They're less concerned with the daily operations needed to deliver the service; that is the role of a service manager.

Quality Manager: A quality manager ensures that a release meets stipulated standards. They may have release managers reporting to them.

Key Concepts in Release Management

Release management is sometimes described as a super discipline, because it involves overseeing several interconnected but distinct disciplines, notes configuration management expert Salman Khwaja (https://www.cmcrossroads.com/article/what-release-management-and-why-it-needed).

One of the central practices in release management is *code management*, which is the process of managing changes to computer code. Using code modules or collections of lines of code, code management simplifies and speeds up the act of making changes to code as well as other code-related activities, like maintenance and debugging. One type of code management that constitutes an important aspect of release management is *version control*. This refers to the management of code for different releases of the same software for the purposes of comparison and reference. Code management follows many of the principles of records management (/record-management).

To perform their role effectively, a release manager must foster facilitation and collaboration, mainly between teams but also within teams. Release managers are typically experienced professionals themselves, so they may be called upon to pitch in.

Processes and policies are critical to effective release management. Problems are bound to arise, and, when they do, there must be protocols to address them. The release manager also functions as a gatekeeper or guardian of the production code. They know every time a code artifact moves out of the organization.

Over a number of releases, it becomes possible to measure for productivity and throughput metrics so that the efficiency of release management becomes visible. Common metrics (/best-practices-guide-agile-planning-project-managers) include such things as velocity and burndown rate.

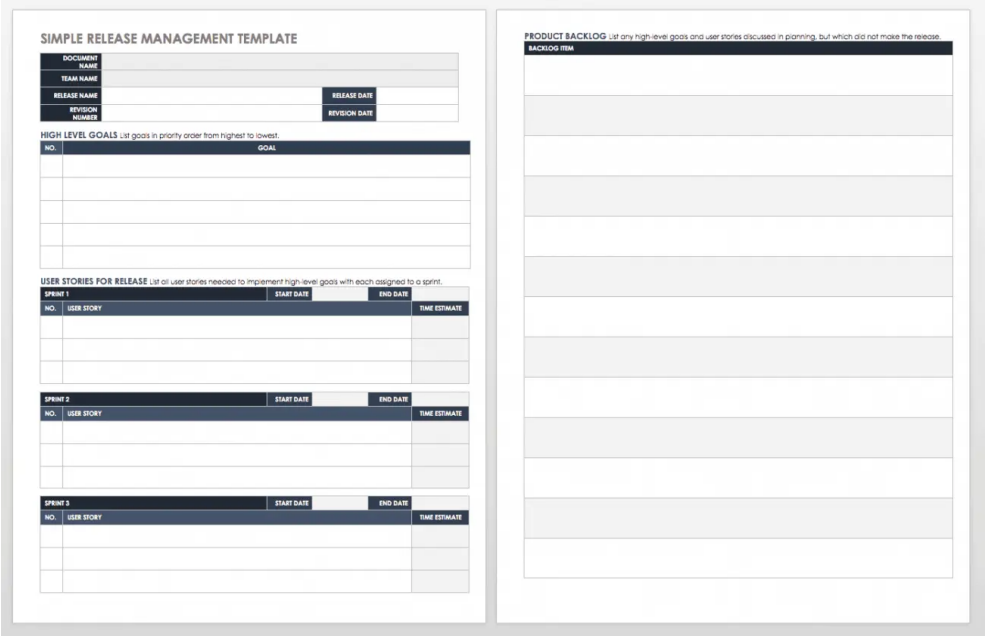
The Cycle of Release Management

Release management typically follows a clearly defined path. It begins with the creation of an organization-wide release schedule. Some companies may release software updates continuously, but others prefer a set schedule. Next comes the scheduling of the product or solution release.

Release management usually begins at the first stage of the development cycle, when the release manager

receives requests for changes or new features. Upon approval of the request, the team designs the new release and planning begins. The developers build the new software. Post-development, the release enters testing, and the team may make modifications before the release is accepted. Thereafter, the release moves into deployment, where it is made available for full use. The post-deployment stage is support, in which bugs or user stories that the team documents here may be fed back into the development cycle as material for a new release.

Simple Release Plan Template


SIMPLE RELEASE MANAGEMENT TEMPLATE

DOCUMENT NAME			
TEAM NAME			
RELEASE NAME		RELEASE DATE	
REVISION NUMBER		REVISION DATE	

HIGH LEVEL GOALS List goals in priority order from highest to lowest.

NO.	GOAL

USER STORIES FOR RELEASE List all user stories needed to implement high-level goals with each assigned to a sprint.

SPRINT 1	START DATE	END DATE	
NO.	USER STORY		TIME ESTIMATE

SPRINT 2	START DATE	END DATE	
NO.	USER STORY		TIME ESTIMATE

SPRINT 3	START DATE	END DATE	
NO.	USER STORY		TIME ESTIMATE

PRODUCT BACKLOG List any high-level goals and user stories discussed in planning, but which did not make the release.

BACKLOG ITEM

Use the template below for a simple release plan. For especially complex and high-budget projects, the release plan framework developed by the Center for Medicaid Services, which can be downloaded here (https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/ReleasePlan.docx), is a good reference.

Download Simple Release Plan Template — Word (/file/ic-simple-release-management-9281worddotx)

Challenges That Release Management Can Help Address

Historically, many companies lacked processes to manage the release cycle. However, because problems can represent a blow to the bottom line, with bugs and crashes costing a lot of time and money (as well as reliability and reputation), release management arose as a discipline to address some common challenges.

Some organizations struggle with deploying code from development to testing and production environments, and while this may have less of an impact on the bottom line, it's no less a cause for concern for the operations staff.

At other times, an organization's internal structure may give rise to conflicts among teams, and release management can decrease friction. Likewise, when there are many delivery teams working in tandem, the risk grows that one team's work will disrupt another's. An efficient release management process can help organizations avoid a number of problems.

Release Management and DevOps in an Agile World

Organizations that base their software development on popular Agile principles (/understanding-agile-software-development-lifecycle-and-process-workflow) typically produce more frequent releases. The Agile approach to software release is called *continuous delivery (/continuous-software-development)*, a method that aims to create code that is ready for deployment at any time. It almost completely eliminates the conventional stages of integration and testing and automates the release process on very short cycles.

Software product development expert and author Jon Quigley of Value Transformation (https://www.valuetransform.com/) notes that studies have shown an inverse relationship between project size and success. He also says research suggests that customers do not use 45 percent of developed features. These statistics argue for small, focused releases that prioritize features highly valued by customers. "The upshot of all of this has driven deliveries of software or development of increments of software that are small, the smaller the better," he says.

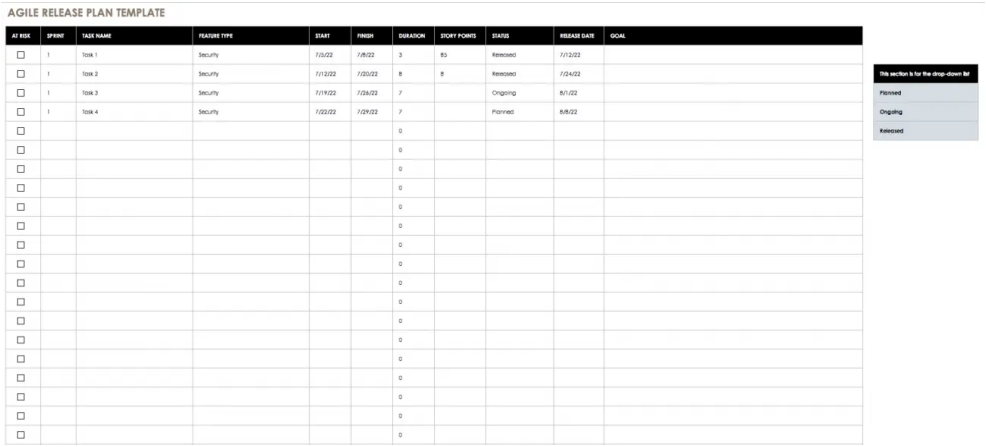
In continuous delivery, release management remains the critical connector between development and production. Release management verifies the integrity of code and makes sure that it functions as planned. Agile methods break down silos often seen with Waterfall methodologies (/when-choose-waterfall-project-management-over-agile), but Agile requires thorough documentation, so processes are clear.

Boeing's Gutierrez says there is a lot of confusion around continuous delivery and another popular practice, *continuous deployment*. "Continuous deployment is the concept that every change made in the code base will be deployed almost immediately to production if the results of the pipeline are successful. Continuous delivery is the concept that every change to the code base goes through the pipeline up to the point of deploying to non-production environments. The team finds and addresses issues immediately, not later, when they plan to release the code base. The code base is always at a quality level that's safe for release. When to release the code base to production is decided by the business," he explains.

To effectively and efficiently release the code base to production, release managers rely on three things: automation, a DevOps mindset, and *continuous integration*. Automation relates mainly to testing functions, while the DevOps mindset vastly improves coordination between development and operations (the delivery and infrastructure people may be the same) in order to smooth out the potentially abrupt transition from the former to the latter.

Continuous integration, meanwhile, is the practice of developers frequently updating their own working copies of code to a mainline — usually about once a day. Continuous integration relies on a stringent system of automated testing, and it tends to weed out errors quickly and speed up the change process.

Agile Release Plan Template


AGILE RELEASE PLAN TEMPLATE

AT RISK	SPRINT	TASK NAME	FEATURE TYPE	START	FINISH	DURATION	STORY POINTS	STATUS	RELEASE DATE	GOAL
☐	1	Task 1	Security	7/5/22	7/8/22	3	85	Released	7/12/22	
☐	1	Task 2	Security	7/12/22	7/20/22	8	8	Released	7/24/22	
☐	1	Task 3	Security	7/19/22	7/26/22	7		Ongoing	8/1/22	
☐	1	Task 4	Security	7/22/22	7/29/22	7		Planned	8/8/22	
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				
☐						0				

This section is for the drop-down list

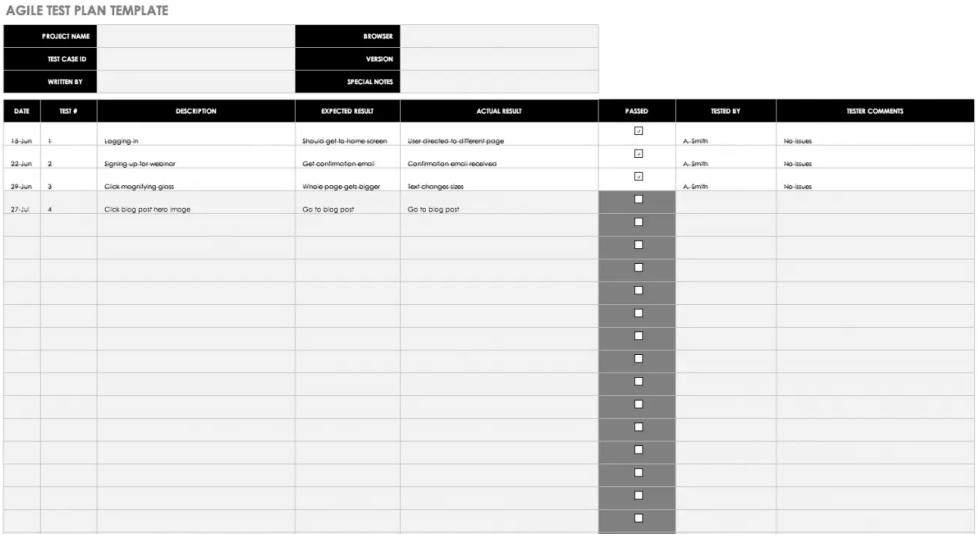
Planned

Ongoing

Released

Agile release planning occurs before the first sprint, so you can focus on the release goals, features, and allocating resources. This Agile release template allows you to list all your tasks, assign each task to a sprint, and calculate the duration based on start and end dates. You can also indicate the status of each task from the drop-down menu and define each corresponding goal.

Agile Test Plan Template


AGILE TEST PLAN TEMPLATE

PROJECT NAME		BROWSER	
TEST CASE ID		VERSION	
WRITTEN BY		SPECIAL NOTES	

DATE	TEST #	DESCRIPTION	EXPECTED RESULT	ACTUAL RESULT	PASSED	TESTED BY	TESTER COMMENTS
13-Jun	1	Logging in	Should get to home screen	User directed to different page	☑	A. Smith	No issues
22-Jun	2	Signing up for webinar	Get confirmation email	Confirmation email received	☑	A. Smith	No issues
29-Jun	3	Click magnifying glass	Whole page gets bigger	Text changes sizes	☑	A. Smith	No issues
27-Jul	4	Click blog post hero image	Go to blog post	Go to blog post	☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		
					☐		

In Agile projects, the testing phase is a dynamic, iterative process meant to ensure that you use the most up-to-date information to define tests and to avoid any scope misunderstandings. This test plan template provides space for you to track actions, expected results, actual results, and whether the test passed or failed.

Download Agile Test Plan Template — Excel (/file/ic-agile-test-plan-template-excel-9281xlsx)

Release Management with the IT Infrastructure Library/IT Service Management

IT service management is a superset of activities that organizations use to design, plan, deliver, operate, and control the IT services that they offer. Among these is a detailed set of practices known as the Information Technology Infrastructure Library. In organizations that manage their IT operations using ITSM, and specifically ITIL, release management is guided by ITIL principles.

In its current form, known as ITIL 2011, ITIL consists of five volumes. Release management is included in the third of these volumes, the service transition volume, which concerns the delivery of services to operational use. As you might guess, the release and deployment management process aims to "plan, schedule, and control the movement of releases to test and live environments." Change management is also included in the third volume.

Releases in ITIL organizations occur far less frequently than in Agile organizations. Release processes in ITIL are managed using ITSM ticketing systems, and there's not as much emphasis on automated release processes. That's not to say that they're any less effective. Organizations that implement ITIL practices can do much more than increase the efficiency of release processes; they can also realize financial gains and increase the business value of the services they offer.

Enterprise Release Management

Enterprise release management (ERM) manages the big picture of releases at the organizational level of entities with large or complex software, such as healthcare systems, big companies, universities, and the like. ERM deals with the coordination of individual software releases and how they fit into the organization's larger plans, strategies, and calendar.

Why is this important? Imagine an organization that develops complex systems of software. Multiple development groups work on different components of these large systems. This structure makes sense internally because it allows developers to specialize, increase focus, and build components piece by piece. But, ultimately, the pieces must converge in a single, seamlessly integrated system. Without overarching release management at the enterprise level, the organization's IT portfolio would lack cohesion.

ERM enables organizations to roll out large software products that work well as an integrated whole, and it helps them to do this efficiently. ERM efforts typically call for many release managers to work in tandem, so they can synchronize their respective releases.

Release Management and the PMI Key Areas of Knowledge

In a 2000 paper (https://www.pmi.org/learning/library/software-release-management-documentation-testing-8901) presented at the Project Management Institute Annual Seminars and Symposium, Franck Aguilh discussed the emerging realization among software development companies that there was a need for a specialized discipline that focuses exclusively on managing releases. The release manager rather than the project manager would play this role, as the project manager has much more expansive priorities.

Aguilh said that successful release management is a process that attains four main objectives: deploying on time, deploying on budget, having negligible impact on existing customers, and meeting the requirements of new customers, all while keeping in mind competitive pressures and technological advancements. He also described release management as a process involving several "major tasks" that are to be performed by distinct "organizations."

Release management calls for coordination between many departments within an organization, each of which fulfills a specialized role. Aguilh's "organizations" stretch from sales and marketing to R&D, operations, and systems engineering to customer support and legal.

"A successful release requires cross-departmental synchronicity," says Dunbeck of BitTitan. "Developers must check code in the right places. QA must perform checks. Operations must manage branches and get the build ready for deployment. But those are just the technical components. If you don't have a way to communicate the change internally, your salespeople will be surprised when showing customers demos that have feature or button changes. Your support team will be unable to answer questions that come in, and your customers will not get the answers they need if you haven't communicated any changes to them via release notes, help center articles, or the like," she emphasizes.

"The easiest way to resolve these issues is to clearly identify your stakeholders for the release and engage them frequently throughout the project cycle. Each organization has different communication preferences. Meetings, a chat channel, a Wiki page, or just email are all good ways to keep people in the loop. The important part is to identify a cadence and then stick with it," Dunbeck concludes.

According to Aguilh's vision, release management rests in part on applying project management principles to software releases. These include the main project management process areas: initiating, planning, executing, controlling, and closing.

However, Aguilh also posited nine processes that are specific to release management. Following are descriptions of these nine processes as well as the order in which they occur:

- **Functional Product Request (FPR):** This is the mechanism for a "functional group" to formalize its request for new features or functions to be added to software.
- **Documentation Process:** The team puts information-sharing protocols for the upcoming release into place.
- **Release Packaging Process:** During this process, the team makes a decision about what goes into the final release, based on factors such as market pressure, cost, revenue, development time, and integration.
- **Development Process/Change Control Process:** These two steps run in tandem, and they're self-explanatory. During development, quality gates may be used to indicate milestones in the development cycle. Change control continues into lab and field tests, which check both whether new functionality is working as expected and whether it has affected existing features.
- **Training Process:** This involves training technical support staff, direct sales and marketing personnel, and telemarketing staff.
- **Customer Testing:** This step happens via beta releases to customers.
- **Customer Notification Process:** The last stage prior to deployment involves informing customers that a release is forthcoming.
- **Deployment:** Deployment itself is a multi-stage process that includes pre-deployment, actual deployment, and post-deployment.

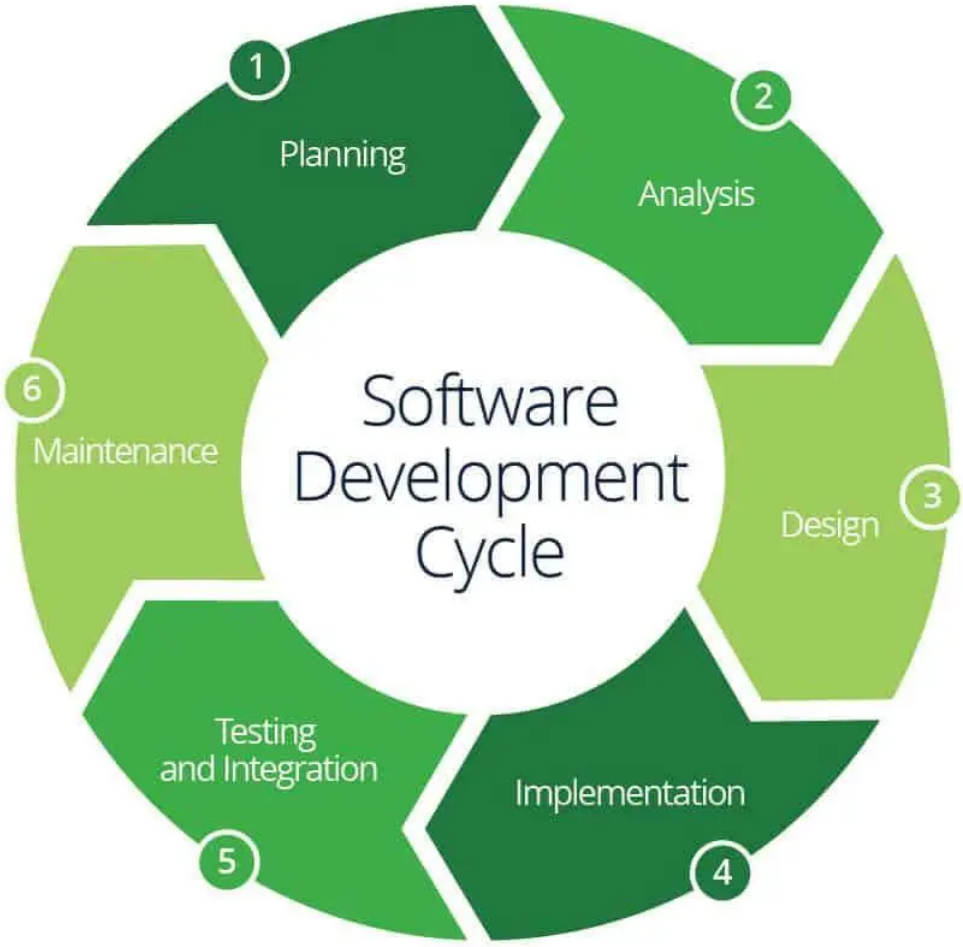
Aguilh suggested that the processes of release management correspond quite closely to those of project management. For example, FPR and release packaging are basically a consideration of scope and planning, while quality corresponds to documentation, development, change control, and training. Executing and controlling, in turn, correspond to training, customer testing, customer notification, and deployment. Closing, of course, maps to deployment. It becomes evident, therefore, that useful parallels can be drawn between project management and release management and that an effective release manager needs to possess the same skills as a project manager.

Release Management Processes by Stages in the Software Development Life Cycle

You have seen how release management processes align with core project management processes. Now, let's look at how release management processes line up with stages in the software development life cycle (SDLC). There are typically six phases in the SDLC:

1. Requirement gathering and analysis

2. Design
3. Implementation or coding
4. Testing
5. Deployment
6. Maintenance



During development, managers lay out a *release policy,* a document that defines the scope, principles, and end goals for the release management process. It uses the organization's strategic objectives to inform and guide the release management process. To see an example, take a look at the release policies (http://www.apache.org/legal/release-policy.html) of the Apache Software Foundation, which promotes open source software projects for public good.

Release managers draw up *release plans* based on the release policy. These plans are broad guidelines for deploying multiple releases.

During the design phase, the release manager makes sure the hardware and software assets that will support the release are designed and configured. Then, coding begins.

During implementation, the developers build and configure the code. During testing, testers try out the code in an operational environment. During deployment, the staff rolls out a live version of the software, and the quality assurance team conducts a quality review to see that the release meets its stipulated requirements. If the release passes the quality review, it is validated and slated for production; this seal of approval is referred to as *release accepted.* If bugs remain, the team will reject the release. An accepted release has a rollout plan that covers the details of deployment, and the company informs clients and end users about the upcoming release. Training may be necessary.

The release units are deployed to production for full launch. Some organizations choose to verify the implementation at this stage in order to ascertain whether the release will run smoothly when live.

In the maintenance stage, developers conduct a review of the release and log issues to be resolved for the next release.

Release Management Schedule Template

RELEASE MANAGEMENT SCHEDULE TEMPLATE

RELEASE NO.	RELEASE TYPE	DESCRIPTION	APPROVAL STATUS	PERFORMED BY	PLANNED START DATE	PLANNED END DATE	RELEASE DATE

To make the timeline clear for all your release activities, create a release management schedule. Use this template to track key deliverables and deadlines to keep everyone on the same page.

Download Release Management Schedule Template

Excel (/file/ic-release-management-schedule-9281xlsx) | Word (/file/ic-release-management-schedule-9281worddotx) | PDF (/file/ic-release-management-schedule-9281pdfpdf)

What's in a Name: Deploy, Release, and Ship

The term *deploy* is often used interchangeably with the words release and ship, but they don't mean exactly the same thing. Deployment represents the installation or running of a new version of code on a server — going "live" with the software. This deployment may occur on a testing server rather than a production server. A release is the software form of this new version. It is intended for an audience beyond developers, either others in the organization or customers. A release generally carries a version number. When a customer deploys a release, it is generally described as an installation of the software. Shipping describes the process of getting code through the entire building, testing, and deployment loop.

Deployment is a crucial juncture in both the software development life cycle and the release management cycle. Deployment marks the moment when the new software version is available for use and the new release goes live. In the words of Paul Jackson at the University of Edinburgh School of Informatics (http://homepages.inf.ed.ac.uk/pbj/), deployment is "getting software out of the hands of the developers and into the hands of the users." For this stage, end users may need training.

Jackson notes that deployment is often problematic: More than 50 percent of commissioned software is not used, mostly because it fails at deployment. If software runs into problems at deployment, developers will "roll back" to the previous version until the issues are resolved.

Jackson also reports that 80 percent of the spending on commissioned software occurs at and after deployment. Of course, this doesn't necessarily mean that the deployment itself is a failure; it may simply be that problems in development only surface at deployment.

But sometimes, deployment is the problem. Maybe, the customer simply isn't prepared for a change in the product experience, as large system releases often demand. Lack of software adoption can also result from a number of other factors: The customer's hardware can't handle the new release; the customer has trouble installing the software; lack of training presents an obstacle; or the new installation "breaks" the customer's other, pre-existing software.

Many of the problems directly associated with deployment can be solved using automation and training, so the end user doesn't have to think about things like system compatibility. Relying on best practices in release management makes these solutions standard operating procedure.

What Is a Release Deployment Process?

The release deployment process is focused on making the software operational in a live environment. For this to occur, the software must go through testing and be officially accepted by the product owner or another business stakeholder. During the deployment process, users receive training on the software update, and team members conduct an assessment or review of how it is performing and how deployment went.

Best Practices in Release Management

In release management, best practices are the guidelines created and refined by companies that have already implemented ITIL with success. The guidelines are owned and published by the British Cabinet Office. ITIL release management processes have been used with good outcomes by space programs, health services, banks, and entertainment companies. The guidelines must be modified by organizations to meet their own requirements and capacities. Remember, though: These are a starting framework, not a gospel.

Although release management matters most during the transition from development to production, it starts with the planning of releases during development. For example, it's a good idea to establish a change advisory board that oversees change throughout the organization. It's also wise to design a single release management system that can be used throughout the SDLC.

Similarly, creating a release process checklist encourages transparency as well as shared understanding of the release management process and how it creates business value. Supplementing with Atul Gawande's book Checklist Manifesto, using a checklist also makes it easier to exert control over the release process, especially if there are metrics to monitor. In addition, it helps to have an active senior sponsor.

"The most common pitfall with release management is not following a release checklist," says Chris Harding (https://www.linkedin.com/in/cmharding), a Release Manager at Microsoft. "The details in a release checklist are key. You may be having an off day, or something may distract you while you're configuring a product, but the checklist never lies," he points out.

Harding believes that a simple error that would have been covered by a checklist was behind Walmart Canada's release of several video game titles on its website in May 2018 in advance of their official debut at an industry conference. "The timing was probably set incorrectly for the product pages, and nobody caught it before pushing to the public. In the medical community, checklists save lives. Release management isn't as dramatic, but it can probably save your job."

Software Release Checklist Template

To keep track of all your release activities, consider using a checklist. This template is available in Microsoft Word and Excel and PDF formats, and includes space to list notes and status about marketing, product development, QA, engineering/DevOps, user experience, technical support, services, and legal tasks. Edit the form to reflect the needs of your project.

Download Software Release Checklist Template

Word (/file/ic-release-management-checklist-9281worddotx) | Excel (/file/ic-release-management-checklist-9281xlsx) | PDF (/file/ic-release-management-checklist-9281pdfpdf)

Some best practices are really just common sense. Continuous integration, for example, improves the quality of the eventual release by catching errors quickly. And, it's not a good idea to launch a release on a Friday, as that leaves no time during the work week for troubleshooting if something does break. It's also worth scheduling a release during a period of low traffic — you'll have to know when your customers are active — so you can do better damage control if things go wrong. Another option is a staged rollout in which features are made available only to a small number of users at a time. And, of course, databases need to be backed up before releases.

Lastly, remember that implementing a release management process is a process in itself. Release management can be iterated and automation achieved a little bit at a time. Don't try to force the process.

Software Tools for Release Management

Software tools can expedite and facilitate the release management process.

For example, contemporary software products are built to support a few platforms, from internet browsers to application servers and operating systems. So, they need to be tested in these environments prior to release. Creating and maintaining all the environments needed for testing poses a problem. But, there's an easy fix: using virtualized and cloud services, such as Selenium and SmartBear, which simplify the process of deploying configurations for the target platforms by creating a simulated environment of each for testing.

"A common challenge is automating the release but not having good automated testing in place," notes Brian White (https://www.linkedin.com/in/bwhite1), Senior Solutions Architect at custom software developer Small Footprint. "If you don't have confidence in the quality of your release, it doesn't matter how fast it can be deployed if it's always going to break when someone starts using it," he says.

To ease the entire release management process, software development platforms and release management systems, such as GitHub, Jira, and many others, offer tight integration of the components needed for a successful release. These can be used throughout the SDLC, from release planning to production.

Such tools will typically be scalable, based on the organization's needs, and they'll centralize update and administration functions, which is especially useful when they're accessible over the internet as SaaS (software as a service) tools. The best tools will also support the testing and deployment of management components, along with change management and integrations with third-party service management components, which allow for an end-to-end delivery system. Look for tools that comply with the IT Service Management Forum ITIL V3 guidelines.

Tools that allow for at least partial automation of the release process are a huge help. It may be possible to automate software delivery all the way to production or to set up a semi-automated series of processes with approvals and on-demand deployments.

Other useful features include the ability to track changes across versions, so it's easy to get to the root of problems and have a structure that takes the pain out of rolling back to the last working version. Logging application feedback, a capability which can be moved to a backlog for upcoming versions, is also helpful.

In addition, consider tools that can support continuous integration and continuous deployment. Continuous deployment goes one step beyond continuous delivery to immediately deploy changes as they pass through the production pipeline. This approach accelerates the feedback process and vastly improves efficiency.

Release Management Metrics and KPIs

Once a release management process has matured, a release manager can turn their attention to performance metrics with the aim of improving the process. Let's look at a few:

- **Release Downtime:** This is the first important metric. Any software release runs the risk of downtime, the length of which may range anywhere from a few minutes to a number of hours or even days. Downtime isn't bad per se; it's often a necessity. That said, it's important to be able to differentiate between the different types of downtime. Estimated downtime, for example, is downtime that is necessary and built into the release schedule, and it's a good idea to compare this with actual downtime, which is measured post-release. There's also unplanned downtime, which over a few releases may be a sign of root problems going unfixed.
- **The Type and Priority of Releases:** Each release is categorized as major, minor, or somewhere in between and is assigned a priority: high, medium, or low. Over a number of releases, look at a breakdown of your releases by type and priority. There should be a good mix of releases by both size and priority. Too many major releases for the same product may indicate serious problems with quality assurance, while having too many high-priority releases suggests that your team is stuck in permanent fire-fighting mode, which tends to burn everyone out.
- **The Number of On-Time Releases:** Since on-time release is not assured, it's as much about release delays. There are quite a few interesting nuggets of information to be gleaned. For example, do some development teams have more late or on-time releases than others? Is there a pattern of late or on-time delivery based on release prioritization? What are the root causes of repeatedly late releases?

You can track some interesting metrics at the enterprise level as well. These include the size of enterprise releases from three different perspectives: projects, system deployments, and features. Another useful set of metrics tracks the numbers of projects, systems, and features de-scoped. (*De-scoping* is when something slated for release fails to pass its release gate and is therefore taken out of the enterprise release so as not to hold it up.)

Case Studies in Release Management

We have covered a lot of theory so far, and you may be wondering how release management works in the real world. Microsoft is a great place to look.

Microsoft's Core Services Engineering (CSE) relies on a combination of continuous integration, continuous delivery, and a cloud-hosted platform named Visual Studio Team Services (VSTS). Their approach to development, which they've termed "modern engineering," is based on the principles of both Agile development and DevOps.

Look at how Microsoft structures its development around continuous delivery, and you'll understand how it benefits so much from better release management. Microsoft software engineers focus on creating a minimally viable product, which draws early adopters while allowing developers to collect feedback for later releases.

Microsoft developers can try code in any deployment environment at any time, and they use componentization to make the code easier to build. Under Microsoft's release management process, they can start unattended builds and unattended deployment that take only a few minutes, without competing for resources. Plus, their deployment validation process is automated.

CSE also embraces the Agile principles of iterative design and rapid prototyping, while using automation to slash testing times. This means the CSE can speed up its release cycles, correct issues quickly, and deliver continuous updates and enhancements in smaller chunks — the last of which also reduces risk, since there's less time spent developing each feature.

And, it's not just tech companies that benefit from release management. Between 2008 and 2010, the IT division of the Central Securities Depository of Turkey (https://waset.org/publications/9999440/release-management-with-continuous-delivery-a-case-study) undertook extensive automation of configuration management, dependency management, infrastructure management, change management, database management, and application configuration management to "completely remove" problems with releases, shorten release time, and radically decrease deployment-related errors.

At United Airlines, a four-step model for release comprising planning, coordination, execution, and automation — all informed by extensive data — helped the company move from spreadsheet-centered release management to a centralized dashboard release tracking and management system.

How to Improve Release Management

Improving your organization's release management capabilities requires understanding the current state of your release management process. You should do this in two ways: quantitatively and qualitatively.

Quantitatively, you should gather a few basic metrics, including average release times, type and priority of releases, number of errors, and number of delayed releases. These serve both to identify the current state of release management and to figure out performance baselines.

Qualitatively, speak to the people who are part of the release management process, especially where development interfaces with operations, and see what they think. They'll be able to point out realities that don't show up in the numbers.

You can get a handle on release management by establishing a regular release cycle, which helps build consistency. (This isn't going to be possible for all releases, of course, but you can do it for large, pre-planned ones.) Put lightweight release processes in place instead of trying to develop a culture from the outset. This allows you to

establish the infrastructure for releases early on and to test and revise it if necessary.

Over time, the processes that work well will become standard practice. Based on your initial survey of release management, you may now be able to start imposing stricter quality requirements and improve on efficiency benchmarks. You can minimize the impact of releases on your users by eliminating downtime and testing for regressions. At this stage, you can also start thinking about regularizing and automating processes, such as testing and verification.

A truly collaborative culture of release takes time to grow, and it needs a release infrastructure to grow around. You can foster this culture by making investments in your staff and investing in release management tools and techniques that encourage people to take a holistic view of the entire release management process.

Improving communication and access to information will cultivate coordination. Let people know that releases will get better, because setting positive expectations matters. "One challenge that teams face is when they view releases as a purely technical hurdle and they forget about good communication within the company and with stakeholders. To overcome this, make sure that everyone on the team understands the goals and risks, and be sure to communicate changes to end users in order to avoid surprises," says White of Small Footprint.

The Release Manager's Job

If shepherding developers and operations toward an end goal and a production deadline sounds like fun, consider becoming a release manager.

A release manager must navigate a complex landscape of projects, development methods, infrastructure, and stakeholders, while coordinating efforts among everyone who contributes to an enterprise release. Release managers must also report to top managers in the organization, all the way up to the CEO (though for more mature organizations, release management is likely to be a more autonomous function), and they take much of the flak when a release goes awry.

Harding of Microsoft says it's important for release managers to have broad project management skills. "Providing other project stakeholders with training and access is key to reducing reliance on an individual. Release managers generally have a pretty specific set of problem-solving skills, but they can lean on other PMs to own elements of product configuration and process," he explains.

So, what are the qualities of a good release manager? They're intimately familiar with the workings of development and operations — that is, both the technical workings and the way people work together. They possess excellent management skills when it comes to work, time, and especially people. They're committed to quality, efficiency, and process. They're advocates for automation, and, on the technical side, they must be proficient operators of source code repositories. And, they're fluent in the myriad inter-project, inter-function, and inter-departmental dependencies that define an enterprise release.

They also know what to believe (and what not to) about release management.

Myths of Software Release and Release Management

There's a lot of conventional wisdom floating around about release management, but how much of it is worth its salt? Luckily for us, Slinger Jansen and Sjaak Brinkkemper (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.6572&rep=rep1&type=pdf), two researchers from Utrecht University in the Netherlands, have already tried (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.6572&rep=rep1&type=pdf)to answer (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.6572&rep=rep1&type=pdf)this (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.6572&rep=rep1&type=pdf)question by comparing cost and value. They found that many myths and misconceptions prevail.

Myth 1: Customers Want to Stay up to Date. In reality, customers only care about updates when they provide useful functionality. If a release doesn't make their experience better, they don't care.

Myth 2: Customers Must (from the Vendor's Perspective) Stay up to Date. This is a case of vendors caring more than customers do about customers using the latest version of software. As we already noted, customers update when they want to and remain happy using older versions otherwise.

Myth 3: The Newest Release Is Always the Best. Depending on a customer's use case, an old (or ancient) release might work just fine, whereas a new one could require them to spend time learning a new interface or new features.

Myth 4: Fixes Can Wait until the Next Major Release. Unfortunately, "the next major release" is rarely on time. And, until it's finally out, the issues will remain out there, plaguing customers.

Myth 5: Workarounds Must Be Avoided at All Costs. Sometimes, people don't mind simple workarounds when the alternative is an expensive, time-consuming upgrade to a new release.

Myth 6: Customers Always Want New Features. If the new features get in the way of their established, comfortable workflows, they don't want them.

Myth 7: Releasing Often Is Bad. As long as you don't release to your external customers each time, frequent releases are not harmful. Frequent releases can shorten the feedback cycle, but confine most of them to internal users and pilot customers.

Myth 8: A Quiet Customer Is a Happy Customer. It's the customers who contact support most often during the early stages of use that prove to be the most content with the product. Reach out to your customers.

Myth 9: Customers Read Release Notes. They don't, of course. They do if they must — that is, if they're limited by choice and are selecting professional software — but they much prefer targeted information that summarizes what's necessary for them, as a particular class of customer, to know.

Myth 10: Having Many Different Releases out in the Field Is Bad. This may be an issue if the vendor provides ongoing support for customers using older versions, but even then, it's a numbers game. Having a small number of

customers on the older version isn't an unmanageable problem.

Big Trends in the Release Management Space

Agile practices and automation are gradually becoming core tenets of release management, and for good reason. Automation shortens release times and can substantially reduce the risk of error. Agile practices push for adopting build automation, test automation, deployment automation, and feedback automation, all of which reduce the burden on the development team.

"Both automation and the use of virtualized/cloud platforms enable and support another hot trend in release management: continuous delivery," says Gabriel Gutierrez, a Release Manager at Boeing. "With continuous delivery, each new software revision is automatically built, tested, and prepared for deployment, turning ideas into reality fast. Continuous delivery allows for more frequent deployments, which means more frequent feedback from the customer. As a result, there is less time wasted by learning quickly if you're making the changes your customers need and care about," he adds.

Another trend that has gained momentum is using distributed version control systems (DVCS), such as GIT, Mercurial, and Perforce, which fundamentally change the way teams collaborate. DVCS allows users to keep self-contained repositories on local computers, complete with full-version histories. This capability makes it possible to work offline and commit changes only to the local setup. Developers can make changes without affecting the central code, which is great for experimentation.

It's believed that the DevOps culture and practice will go mainstream at larger companies, with automation of all stages an achievable target. Startups, meanwhile, will likely start adopting the DevOps methodology from the get-go. This is in part because DevOps, done properly, means fewer failure times, faster recovery rates, and a greater number of deployments, which, in turn, means a decrease in (very costly) downtime. As DevOps adoption increases, so will the process of *shifting left*, which is when automated testing and performance monitoring happen earlier in the life cycle and security verification becomes part of the delivery pipeline.

"While not a new trend, more frequent and rapid deployments have gone from a technical challenge to an expectation from stakeholders," adds White of Small Footprint. "As stakeholders are more informed about the possibilities and benefits of more frequent deployments, they are no longer satisfied with waiting weeks between releases and are pushing development teams to release more often. If the team doesn't have a solid continuous integration/continuous delivery infrastructure in place, it can put huge stress on the team," he concludes.

Quigley says that successful execution of these rapid and well-defined increments of software content requires "a level of diligence concerning configuration management and the tracking of a product's growth over time, since there are more iterations delivered to the customer."

Further Resources on Release Management

For a more technical introduction to release management, check out this white paper (https://www.kn-portal.com/fileadmin/xxx/AgileReleaseManagement-whitePaper.pdf) by Jez Humble of ThoughtWorks Studios. And, once you're done with that, Electric Cloud has a vast array of release management resources (https://electric-cloud.com/wiki/display/releasemanagement/Release+Management+Process) that delve into specifics.

If you're thinking about release management as a profession, and you're already working on Agile, check out this short course (https://www.pluralsight.com/courses/agile-release-management) from Jan-Erik Sandberg on Pluralsight. And, if you think getting certified in ITIL is a good idea, Pink Elephant offers official ITIL certification courses (https://www.pinkelephant.com/en-us/Course/ITIL-Foundation), starting at the Foundation level.

Finally, if you're just looking to get your hands dirty with release management right away, take a look at how Harvard Information Technology (https://huit.harvard.edu/files/huit/files/poster_v4.pdf?m=1389206107) does it.

Improve Release Management with Smartsheet for Software Development

Empower your people to go above and beyond with a flexible platform designed to match the needs of your team — and adapt as those needs change.

The Smartsheet platform makes it easy to plan, capture, manage, and report on work from anywhere, helping your team be more effective and get more done. Report on key metrics and get real-time visibility into work as it happens with roll-up reports, dashboards, and automated workflows built to keep your team connected and informed.

When teams have clarity into the work getting done, there's no telling how much more they can accomplish in the same amount of time. Try Smartsheet for free, today. (https://www.smartsheet.com/try-it?lpv=botlink&trp=6810&lx=&tg=&sc=&cta_from=&cta_to=)