Using Scrum to Develop an Automated Testing and DevOps MVP on a Wing and a Prayer

Introduction

In the third quarter of 2021, a 20-year-old IT project began in earnest to build its first DevOps pipeline and automate as much testing as possible. For the prior 20 years, the majority of its tests were not automated, although a small team of developers did create a small, automated testing MVP in the second half of the project. One of the goals of the project beyond its 20th year of operation was to automate even more tests and eventually build a DevOps pipeline. The project used Scrum in the second half of the project, but once again, the majority of its tests were not automated, labor intensive, no DevOps pipeline existed, and major testing events were few, far between, and extremely laborious and expensive. In an effort to automate more tests, two Scrum teams were formed to get the ball rolling and move the needle forward with respect to automated testing. At this point, few people really understood what DevOps was all about and conflated DevOps and automated testing. All the while, the project itself was in a ramp up phase to create more new software than it had in two decades, and the development teams themselves were expected to validate their Scrum Sprints by creating their own automated development tests. So, when this story picks up, there were two new product development teams creating new software and automated development tests, and two teams dedicated to automate any manual tests created over the prior two decades. There was little governance, oversight, or architectural runway, and the four teams were expected to self-organize and magically create a new DevOps pipeline on a wing and a prayer. One professional Scrummaster was appointed to oversee the two new product development teams and a professional Agile Coach was appointed to be the Scrummaster of the two automated test teams. Both used release planning in addition to the basic Scrum ceremonies to create quarterly plans as well as two-week Sprint plans for the four teams and manage delivery performance. The four Scrum teams had about seven developers each in total, although the two test teams were later combined and enlarged. There were over 35 Scrum teams on this project, some of which were business and other non-development teams, but these four teams served as a microcosm for how all new software development would occur over the next four years of the project. This case study focuses first two DevOps releases of the two automated test teams although all four teams were inseparably joined at the hip, literally.

Release One Backlog Grooming

Although the 20-year-old project had been using Scrum for the last 10 years, a proper Agile Product Management (APM) team had never been established. At this point, it's safe to say this was a hybrid traditional-Scrum project, where traditional project management and business analysis were used to form big up front integrated master schedules, requirements, and designs. A test engineering team existed for the prior 20 years which created a mountain of manual tests that took months, if not years to execute. At this point, the project was in its fifth or sixth major release testifying to how labor intensive its manual test approach had become, which is par for the course for traditional projects. There was a small, automated test team for the last 8 years that built up a small, automated testing MVP consisting of an automated testing platform and about 10 or 20 automated tests which had previously been manual tests. These tests were run twice a day for the last 8 years as a proof-of-concept, but they paled in comparison to the mountain of manual tests that had to be run for every major release. The project management team hoped to reduce lead and cycle times by asking new product development teams to create automated development tests that could be added to the automated testing suite from this point forward. And, of course, the small, automated testing team was asked to begin in earnest to automate as many of the existing manual tests as well. That is, transition their small 8-year-old automated testing MVP into a production suite that automated all old and new tests as quickly as possible. So, from this point forward, the project would burn the candle at both ends to reduce the backlog of manual tests as much as possible (i.e., not create any new manual tests, automate all new tests, and automate all old tests as well). Once again, little advanced planning, analysis, systems engineering, agile product management, nor architectural runway was created to realize the goal of creating a DevOps pipeline. Project management expected everyone to begin automating as many tests as possible, a DevOps pipeline would magically appear, and this would constitute the transition from manual testing to DevOps. This fails to mention that up until this point we're only describing the application layer of this project and the upper application middleware layers of the new DevOps pipeline. That is, this was a hardware intensive global data center project, so eventually DevOps would have to be applied to the hardware and operating infrastructures as well. At this point, we're not talking about these lower layers although project management would later assume the automated testing team's DevOps pipeline would also subsume the infrastructure DevOps needs as well (also on a wing and a prayer). That is, project management naively assumed the small ragtag group of automated testers who built the initial automated testing MVP over the prior 8 years would somehow transform from a caterpillar into a flaming phoenix and DevOps the hell out of their global data center infrastructure without bricks, straw, personnel, budget, schedule, etc. For such a traditionally oriented Scrum hybrid, project management seemed to have a lean concept of planning (i.e., software development teams would absorb the enormous technical debt of creating their own massive architectural runways without additional resources). The existing automated testing team did little up front backlog grooming prior to release planning as the only agile product backlog consisted of verbal instructions to automate as many manual tests as possible. No lists of manual tests were provided in advance for analysis of feasibility and story mapping into agile product roadmaps or possible new experimental MVPs. It wouldn't have taken much effort to charter a makeshift agile product management team from manual and automated testers, analyze a few manual tests, and create a small backlog of features and stories for automation over the next 90 days. However, this never materialized prior to the first release planning event. Because project management never instituted agile product management over the prior 10 years, individual teams simply did not have the skills nor impetus necessary to perform proper backlog grooming prior to release planning events. The new automated testing team was also told to automate as many manual tests as possible prior to release planning, which was formed only a few days before release planning. However, systems engineers were intelligent enough to provide a small backlog of user stories to create a new automated testing MVP (i.e., the component that new product development teams would need to automate their own tests). The two new product development teams were given complex functional requirements in the form of user stories, told to write automated tests to validate their Sprints, and more importantly, come hell or high water, meet split precision EVM cost and schedule targets (i.e., they'd be rewarded for hitting precise EVM scores vs. working code and tests which came back to bite the project and all four teams pretty hard within six months).

Release One Planning

The was the second release planning event of the new project period of performance. However, this was the first release planning event where the two automated test teams and two new product development teams would be required to automate as many tests as possible. Therefore, this release planning event constituted the project's first major push to create a DevOps pipeline to reduce the lead and cycle times associated with its 20-year-old manual testing approach. Once again, marketing promised DevOps would be used to reduce lead and cycle times that was only a marketing buzzword at this time. As far as project management was concerned, the goal was to automate as many tests as possible. In other words, the average person conflated automated testing and DevOps or relegated automated testing to the application layer while reserving DevOps for the global hardware infrastructure which is a common phenomenon throughout industry. That is, software developers associate the term DevOps with automating the application software development ecosystem, whereas IT product platform or data center designers associate the term DevOps with operating and maintaining the global hardware, network, and operating system infrastructure. So, application developers forget the OPS part of DevOps, while IT infrastructure managers forget DEV part of DevOps. Its unclear at this point why DevOps has become almost fully subsumed by IT infrastructure managers. Even application software developers no longer believe the term DevOps applies to them, although it emerged from the software development community. It's even surprising why the project management team believed automated testing was key to application software development success, since many DevOps pipelines focus on pushing out development code to production platforms as quickly as possible in an automated fashion and place little attention on automated application testing at all. However, the project had a long-standing dependency on labor intensive manual testing that needed fixing. The project's test lead made the wise decision to combine release planning for the two automated test teams (i.e., treat them as one team for release planning purposes). The two new product development teams did the same. The release planning goals of the two automated test teams consisted of stabilizing the new automated testing tools which were suspect at this point, establishing a new automated testing MVP for new product development teams, creating new automated tests, and continuing to run existing automated tests. The two new product development teams, of course, focused on goals for creating new application software and creating their own automated testing MVP. In fact, each of the new product development teams created different automated testing MVPs. So, we now had the beginnings of four automated testing MVPs (i.e., the automated testing MVP created over the prior 8 years, the new automated testing MVP that would replace it, and the two independent automated testing MVPs created by new product development teams). It's unclear how the new product development teams got the message to create their own independent automated testing MVPs, since all four teams shared a common business analyst feeding automated testing stories to all four teams. Perhaps, the business analyst felt it was a good risk reduction activity to have all four teams in a space race against one another and see which one got to the moon first. None-the-less, all four teams were now off to the races for the next 90 days on a wing and a prayer, with little understanding of the outcome for six months.

Release One Execution

Once again, this was the second major release of the new project period of performance, but the first major release in which large new quantities of software would be produced over a six-month period, and of course new automated tests were to be created as well. Little thought was given to pre-engineering an automated testing or DevOps MVP architectural runway upon which all four teams could develop their software. The two new product development teams were to create new application software functionality, create their own automated tests to sell off their Sprints, and do this within extremely precise EVM cost and schedule constraints. Again, new product development teams were rewarded for hitting precise EVM targets vs. creating working software which seemed antithetical to the Agile Manifesto, given this project's prior 10-year history using Scrum. Perhaps, the project management team assumed the new product development teams would create working software in addition to hitting precise EVM targets. Even in the traditional iron triangle of scope, time, and cost, usually one of the variables is optimized while sacrificing the other two, so focusing on time and cost, especially time, signaled that little else mattered to project management nor new product development teams. Although the two new product development teams shared a common Scrummaster, who served as one of several proxy product owners, the teams themselves had little in common from an architecture, implementation, or test standpoint that wouldn't emerge for six months. No one said agile product management or creating common architectural runways was easy, whether formally, informally, or ad hoc. All four teams were suboptimizing along with the other 30 or so project teams.

- **Automated Test Team 1**. The new automated test team's goal or focus provided by project management was to automate as many new tests as possible. However, team one was quickly hijacked by other slightly more visionary stakeholders, particularly the chief architect and lead systems engineer, both of whom realized the necessity of creating a new automated testing MVP for all four teams. Unbeknownst to project management, team one focused on creating a new automated testing MVP with user stories handed to them under the table from these two stakeholders. This constituted an informal agile product management backlog, but these passive stakeholders were not the official agile product management team, project management team, nor business analysts. That is, this project had a 10-year history of feeding unvetted user stories to individual application software development teams after release planning (acting as informal product owners). That is, there was a formal project management reporting structure, and like an iceberg, there was an informal technical reporting structure beneath the surface of the water. Technically, Scrummasters reported directly to the formal (above water) project management team, while technical leads or chief programmers reported to the informal proxy product owners behind the scenes or beneath the water. Occasionally, these informal proxy product owners tried to influence Scrummasters, but knew that the most effective way to influence the architecture and implementation of the system was to feed their informal backlogs directly to the technical leads (i.e., chief programmers of each new product development team). So, the technical leads or chief programmers had their own shadow organization or network behind the scenes that was more influential than the formal project management team. Transparency, openness, honesty, integrity, justice, and directness is a major part of the Agile Manifesto, Scrum, and the role of the Scrummaster, who is there to help their teams satisfy their scope, time, cost, and quality constraints (and produce working software at the end of each Sprint). Therefore, there is a clash of worldviews between the world of Scrummasters and shadow reporting structures that exist in all organizations like this one. In other words, shadow proxy product owners know better than to reveal their intentions to Scrummasters (i.e., the notion that technical leads or chief programmers have two backlogs—A formal and an informal one). In fact, technical leads or chief programmers often have a way of hiding informal work behind formal user stories. The new automated testing team spent the first release creating the first version of the new automated testing framework MVP and didn't create a single new automated test as required which disturbed project management.

- **Automated Test Team 2**. The purpose or goal of the second automated test team was to continue operating their small, automated testing MVP created over the prior 8 years. In addition, they were to begin creating new automated tests using the first team's new automated testing MVP. The problem was that it would take 90 days for the new automated testing MVP to complete. There was significant turnover on the second automated testing team throughout the first release. For one, there was a new Scrummaster, the old technical lead or chief programmer who created the 8-year-old automated testing MVP resigned, and a new technical lead was added with little patience nor social skills. Secondly, the project's test director did not provide a product backlog of tests to automate, there was a lot of infighting among the team's veterans throughout the first release, and the rest of the developers were too new or junior to contribute any new value adding work. The newest member was forced to resign, a senior developer also resigned in frustration, and the new technical lead or chief programmer instantly created a culture of fear or lack of psychological safety that paralyzed the team for the next 90 days. This also negatively impacted the Scrummaster, which was par for the course as far as this project was concerned. In other words, the technical leads or chief programmers had a lot of power on this project and were respected and feared by all. This is essentially a computer programming or application software development project, so competent developers are valued very highly, and their needs placed above all else. Once the technical lead felt she had control of the team through fear, then she settled down a bit by the end of the first release. The senior developer who quit created one of the first new automated tests in several years in only a couple of days before leaving. This was at the urging of the new Scrummaster who desperately tried to motivate the second team to automate a new test. It's a bit unclear why the project's test director didn't take a more active role in feeding a product backlog to this team consisting of tests to automate. Is it because he had a vested interest in the manual tests? Is it because he felt the first team should implement the new automated tests and the second team replaced or downsized? Is it because he didn't value the 8-year-old automated testing MVP created by this team which constituted the program's first DevOps pipeline? Was he too focused on planning for major project testing events using the manual tests? Or was there simply too much confusion between the project's technical scope from the prior 10 years and the project's vision for the next 10 years? It was probably a perfect storm, and all of these factors prevented the project's test director from deconflicting the chaos within the second automated testing team. None-the-less, by the end of the first release, the second automated testing team created its first automated test and now had a minor vision for how to proceed in the next release. A major project testing event also took place at the end of the first release, which subsumed the focus of the project's test director and the second testing team.
- **New Product Development Teams**. The two new product development teams were given one directive—Complete your new software on-time and on-budget, especially on-time. Like the two automated testing teams, technical leads or chief programmers were in full-control of those teams that coded night and day—About 50-60 hours per week—For the first 90-day release. They were small teams, and some of their developers were new. One or two programmers on each team created all of the code single handedly—Particularly the chief programmers—While newer members were relegated to minor tasks. Basically, newer developers were kept out of the way of the more senior programmers. If a new developer showed some initiative, then they were tolerated for the most part. However, if a new or existing developer was totally inept, they were immediately removed, which did happen. There was one systems engineer for the two new product development teams along with one Scrummaster. The systems engineer acted as proxy product owner, fed them their user stories, and also fed them the designs of their automated tests. Remember, no common new automated testing MVP existed as of yet, which was being created by the first automated testing team. Therefore, the developers had to create their own automated testing MVPs one user story and one test at a time. Each Sprint, they tried to test and demonstrate as many user stories to their real product owners as possible. However, the developers seemed to create a large backlog of untested code and then sold off all of the user stories at one time at the end of the first release. It's unclear if this coincided with the completion of the first version of the new automated testing framework MVP by the first automated testing team, or if the developers created a large batch of new code and tested it with their own testing MVPs. There is evidence that they were shadowing the work of the first automated testing team, riffing it, and creating their own (shadow) automated testing framework MVP behind the scenes. Needless to say, the new product development teams sold off all of their code by the end release one. The Scrummaster had little control over the technical leads, but simply reported on their progress looking as though he was in full control of new product development teams. However, whatever automated tests the developers were using to sell off their stories to product owners, they were minimal at best, because they were not fully automatable, still involved a lot of human interaction, they were not repeatable, and they could not be run over and over in an automatable Continuous Integration, Continuous Delivery, nor DevOps pipeline. Furthermore, as indicated, each of the two new product development teams were not sharing a common automated testing MVP but evolved their own (minimalistic) automated testing MVPs rather than a common reusable DevOps architectural runway. The technical lead of the first automated testing team believed the new product development teams were using the common new automated testing MVP, but this was simply a leap of faith (as they were only using it as a reference design for custom, redundant, and minimalistic automated testing MVPs). The new product development teams made a lot of progress in the first release, but still had 90 days to complete their code by the end of the release two, and it had to pass a major project test event.

Release One Scrum Ceremonies

After each major quarterly release planning event, Scrum was followed meticulously, including backlog grooming, sprint planning, daily standups, sprint demonstrations, and retrospectives (without skipping a beat). There was also a Scrum of Scrums (SoS) meeting held twice a week for the 35 or so teams on the entire project (whether they were business, technical, administrative, hardware, or software development teams). There were some project teams exempt from release planning and Scrum ceremonies like the project management and systems engineering teams. Of course, this came back to bite the project pretty hard every time as project management and systems engineering never had their work planned nor completed on time. Information from the project management and systems engineering team never reached the 35 Scrum teams in time to do any advanced planning, especially in the form of formal or informal Agile Product Management (APM). Agile coaches urged the project management and systems engineering teams to apply advanced APM, release planning, and Scrum practices throughout the project to no avail. The first few Sprint planning events for the two automated test teams were a bit rough throughout release one. As suggested earlier, each of the 35 teams first created a 90-day release plan each quarter and then immediately went into a two-week Sprint planning session. Each project quarter or release consisted of five two-week Sprints and a sixth three-week Sprint in which release or quarterly planning was performed. On the first business day following the sixth three-week Sprint is when Sprint planning occurred. So, all 35 teams immediately shifted from quarterly release to two-week Sprint planning within 48-72 hours. Two back-to-back planning events was a bit much, but eventually all 35 teams fell into a rhythm and didn't complain as much anymore. Of course, the two planning events had different scopes (i.e., Release Planning was a high-level 90-day strategic roadmap, whereas Sprint Planning was a two-week tactical day-to-day roadmap with up to a 50% expansion ratio per Sprint).

- **Automated Test Team 1**. The first automated test team tended to deviate from the Release Plan and create entirely new Sprint Plans that greatly differed from the 90-day plans. This is because their release plans were rather porous and non-committal in nature, while detailed user stories and backlogs emerged on a day-to-day basis. Each of the developers was assigned a user story and task to create part of the new automated testing framework MVP. They were simple, but effective and highly technical software development task plans that only a computer programmer could appreciate. Again, there was only one Scrummaster who oversaw both automated test teams, and the first team's technical nature was hard to comprehend for the Scrummaster at first. Of course, the technical lead or chief programmer's detailed task plan was only strategic in nature to him, as the real computer programming needs emerged on a daily basis which were performed under the guises of the two-week Sprint plan. So, the Scrummaster had to not only reconcile the differences between the Release and Sprint Plans and explain these in SoSs to the project management team but try to ascertain what was happening below the radar or underneath the Sprint Plans as well. The technical lead wasn't being dishonest or building an entirely different product, but simply building the to-be building blocks of the new automated testing framework MVP one line of code at a time (which was difficult to capture in above the radar Sprint Plans). Eventually their work stabilized and became more predictable, and the technical lead or chief programmer developed Sprint Plans for the first automated test team before Sprint Planning. Although he could do so in about an hour or so, in a makeshift one or two-hour backlog grooming and story mapping session, it was still a labor-intensive process. However, walking into Sprint Planning with a well-groomed story map reduced Sprint Planning from about 1.5 hours to about 30 minutes or so. Sometimes, Sprint Planning extended to 45 minutes just so the software developers could ask questions about the scope of their new user stories and tasks and readjust their descriptions and story points up or down if necessary. The technical lead was in the habit of assigning two-week user stories to each developer who tended to take the entire two-week period to finish their stories. There was enough excess capacity in the user story sizes for administration, discovery, development, code reviews, testing, and demos (for the most part). Parkinson's Law was in full effect and each software developer never finished a user story in under two weeks (most likely procrastinating until the last minute, finishing early but not reporting it, or simply working at a snail's pace). This isn't to say that the first automated testing team was moving slowly, but certainly not as fast as they could have. Remember, project management wanted team one to create new automated tests in release one, but they devoted the entire release to creating the new automated testing framework MVP. The technical lead knew what he was doing, and it was best to keep the first team's focus on one major feature instead of two (limit its WIP). This is especially true since the automated testing framework MVP was new, risky, uncertain, and required a lot of careful attention. Daily standup meetings were a bit longer than usual (i.e., 30 vs. 15 minutes), but this is because developers had many impediments arising from the antiquated technology stack which the technical lead could quickly overcome with impromptu explanations. Sprint reviews often consisted of demonstrating live code associated with the new automated testing framework MVP. Retrospectives were short, sweet, and to the point and rarely focused on how to improve the first automated testing team's performance but rather on the project as a whole. The Scrummaster often translated the technical lead's high-level Sprint Plans into executable plans in the ALM tool, did his best to track the team's work on a daily basis, report it out in SoSs twice a week, and successfully close each user story at the end of each Sprint. The ALM tool itself was a bit complicated and the Scrummaster was sure to fill out all of the mandatory fields, including acceptance criteria and links to EVM budgets. The Scrummaster and technical lead were new to one another and often at odds with one another too, but they managed to hit a rhythm halfway through release one and seemed to get on the same page with respect to the management and operation of the first automated test team using Scrum.
- **Automated Test Team 2**. The second automated test team tended to follow their 90-day Release Plan a little closer than the first automated testing team. So, Sprint Planning consisted of having developers self-select user stories for each Sprint that were created during Release Planning. User stories that no longer applied were invalidated, story points and descriptions were adjusted, some user stories were pushed into the backlog for future consideration, and sometimes new user stories were created. Although the second team followed their Release Plan as closely as possible, this team's Sprint planning often went for 1.5 to 2 hours, just because this team liked to chat, analyze, and groom the backlog as they went along. So, Sprint Planning often doubled as a backlog grooming session as well. This was a little perturbing to the Scrummaster who was accustomed to

short, timeboxed, and by-the-book Scrum ceremonies vs. elongated brainstorming sessions. It seemed as though team two was always in brainstorming mode. The Scrummaster also liked brainstorming sessions so was able to adapt to the cultural style of the second team after some fits and starts. However, eventually the Scrummaster was able to corral the second team into a one-hour Sprint planning session. Daily standups were often held to 15 minutes, unless an important topic emerged, and they needed to devolve into brainstorming sessions (which was often the case). Again, the purpose or goal of team two was to create new automated tests, but the new automated testing framework MVP under construction by team one was yet to be complete. Also, no user stories for specific tests had been created by the test director or any other project constituency. Furthermore, the second team was to create the new tests in a programming language of which they were unfamiliar. There was a ton of fear, uncertainty, and doubt and a lot of infighting. The Scrummaster's first job was to settle the team down, foster several team building activities, and schedule mentoring sessions with each member of the team to settle them down. Of course, all of this had little effect as the second team was melting down faster than the Scrummaster could cool down its intense heat. As mentioned earlier, the prior technical lead or chief programmer responsible for creating the prior 8-year-old automated testing MVP resigned, plunging the second team into utter chaos. It was the Wild West, and it wouldn't settle down until two more people left the team, including one of its most experienced developers. The Scrummaster had experience with problem Scrum teams and knew one method of quickly settling them down was to schedule mob programming sessions, which he immediately did. Mob programming did several things: 1) get them out of individual mode in which the prior technical lead had them; 2) get them to work together as a team which they hadn't done before; 3) build teamwork, rapport, and trust among its members; 4) build confidence so they could continue functioning even without the prior technical lead who was viewed as a demigod; 5) develop the team's first discovery spikes, prototypes, and automated testing components; 6) stop the incessant infighting; and 7) share the pain of development equally with each developer as some felt they were unequally burdened with the difficult tasks. Like magic, this seemed to stem the tide of insurrection pretty quickly, but it didn't solve every problem. The Scrummaster quickly communicated the team's interpersonal difficulties to the test director who empowered the second team to continue using the 8-year-old automated testing MVP and stop waiting for the first team to finish the new automated testing framework MVP. This didn't seem to jumpstart the productivity of the second team as there was no process for formal agile product management, lean UX, story mapping, discovery, business analysis, nor prioritization of existing manual tests. In other words, the second team didn't have a process for quickly analyzing manual tests, decomposing them into user stories and tasks, and automating them very easily. They did have an 8-year-old automated testing framework and about 20 or so automated tests they'd created, but the prior technical lead did all of the thinking for the second team, and they didn't feel confident nor empowered enough to think for themselves and kickstart the process. As mentioned earlier, one of the developers quickly created a new automated test to prove it could be done and then resigned. Up until that point, none of the developers on the second team had been empowered to perform such a feat (i.e., single handedly create a new automated test without being micromanaged the prior technical lead or chief programmer and being second guessed every step of the way). Basically, the prior technical lead created a culture of fear, lack of confidence, and lack of courage in their individual decision-making abilities and technical skills. The Scrummaster spent the entire first release encouraging individual developers to step out of their shells and do something useful without being told what to do or how to do it. The Scrummaster did his best to keep the second team's load as low as possible, keep project management expectations low, and generate as many successes as possible just to keep the lights on. No one was even sure the second team would not be completely replaced by the first automated test team after release one. The second team created unnecessarily detailed Sprint Plans and its demos and retros were a bit choppy.

Release Two Backlog Grooming

With a rather stormy release one behind the teams, release two backlog grooming quickly ensued. Remember, project management and systems engineering didn't apply formal or informal Agile Product Management (APM) nor Scrum ceremonies. So, all 35 teams on this project had to enter a two-day Release Planning event with no product backlog, story maps, well defined features, nor user stories. The teams themselves didn't have the excess capacity or impetus to perform these activities nor create these artifacts in advance of Release Planning either. The Scrummaster of both automated testing teams did his best to schedule backlog grooming sessions with the test director, test engineers, and technical leads or chief programmers of both automated testing teams. It was clear at this point that enough was enough, it was time for the rubber to hit the road, and both teams were on-the-hook to stop making excuses and begin creating new automated tests. Of course, the new automated testing framework MVP created by team one was still in its infancy and would need some more basic features and capabilities in order for new product development teams to use it. The Scrummaster assumed the role of automated testing agile product manager, brought the constituencies together, and began identifying user stories for new automated tests. With some perturbation, the first team identified about five new automated tests they could create in addition to extending the new automated testing MVP. It was a tall order for a group of brand-new developers who had never created a complex automated test before. The second team also identified three new automated tests they could create, including one large one that would take an entire release. No one had time, patience, or capacity to do a deep dive of the new automated tests, properly decompose them into story maps, identify an MVP, and prepare well-groomed release backlogs. There were also major project testing events coming up which would eclipse the resources of both teams in release two as well. They didn't have much capacity for proper Agile Product Management (APM), but this was certainly a start since the project management team failed to establish these practices over the prior 10 years.

Release Two Planning

Release two planning came up quickly, both automated testing teams planned together as they had for release one planning and they were off to the races. There was a lot of perturbation, uncertainty, complexity, and possible risk associated with creating new automated tests, especially for the first team which hadn't done so before. Although they had created their automated testing MVP for new product development teams over the prior 90 days, they were now in a position to eat their own dog food. That is, they would be the primary users of the new automated testing MVP. This would be an opportunity to shake it out, operationalize it, stabilize it, debug it, and extend it with new features if necessary. In other words, the first team would now become its own product owner or source of new user stories with respect to the automated testing framework. The new product development teams were yet to embrace the new automated testing framework MVP. The technical lead or chief programmer was most perturbed with having to blaze a new trail and begin creating new automated tests. The thought of doing this disturbed him greatly, and the test director and Scrummaster did all they could to settle him down. The Scrummaster improvised quickly, which was one of his skills, and suggested how the first team might organize a discovery centric release plan. After munching on it for a day or two, the technical lead improvised himself and created a two-user story system. Each user story constituting a new automated test would be assigned to pair programmers, which was a nice risk reduction technique. The first Sprint would be a discovery Sprint for the pair programmers to analyze the tests, meet with test engineers, run the manual test procedures, setup and configure their development environments, rapidly create preliminary designs, begin establishing architectural runways, and figure out how to eat their own dogfood. The technical lead of team one served as chief designer and developer to mitigate the risks of venturing out into the unknown. In other words, the first Sprint would be an Agile Product Management or Lean UX Sprint and the technical lead would be the Agile Product Manager to create the story maps or backlog of tasks necessary to implement the new automated tests for the remainder of the release. It was a risky strategy given the uncertain complexity of each new automated test, but at least they were willing to try and that was good enough for the Scrummaster. The second team proposed to complete one new low-risk automated test and undertake two highly complex release long automated tests. It was a different approach than the first team who was willing to dive into the deep end of the pool. The second team had more experience than the first creating new automated tests, so they're approach was to chip away at the complex user stories one Sprint at time. If they finished early, they would simply take on another test. The project management team was happy the first team committed to five new automated tests, although they would only begin and complete three of them. However, project management was unhappy the second team only committed to three new automated tests, although they would complete two out of three of them. Remember, the first team was to use the new automated testing framework MVP while the second team was to use the 8-year-old automated testing MVP. Team two didn't learn many new technical skills from team one, but they learned how to manage uncertainty with confidence.

Release Two Execution

Overall, project management was extremely unhappy with the use of two automated testing framework MVPs—A new and an old one—Even though this was a risk reduction technique by the test director to get the ball rolling creating new automated tests. And, of course, the two major new product development teams weren't using either of these two frameworks, and each of those teams had its own minimalistic framework as well. Keep in mind, that it's generous to deem what the new product development teams created, "an automated testing framework," as they used the smallest amount of manual and automated testing necessary to demo their user stories to their product owners. Again, the two new product development teams were pressured by the project management team to hit precise EVM or cost and schedule targets, so creating working testable code seemed to be their least concern. Of course, the test director didn't understand this and began pressuring the automated testing teams to collect the development tests and integrate them into the testing frameworks during release one. It was a noble idea, but something didn't seem right, so the Scrummaster pushed back saying the developers would have to run their own tests. It was a smart move at first since the new development tests were in no shape to be automated early on. There was a major project testing event at the beginning of release two and developers were forced to run their own tests manually, which took over six hours. Worse yet, they corrupted their tests the night before the test event and over half of the new product development tests did not run at all. The automated testing teams dodged a bullet with the crude new product development tests, but not for long. The new product development teams ruined the project's first major test event with their all-night hackathons, but the test director smoothed things over like a professional as he'd done for the prior 10 years. Only the Scrummaster of the automated test teams knew what happened, while the Scrummaster of the new product development teams was heralded as a demigod for hitting precise EVM targets. Eventually, project management began hiring new developers in droves, adding them to the automated testing teams, and combining the two teams into one large 25-person team. The "Mythical Man Month" was in full effect after nearly 50 years, and project management didn't want to add new people to the new product development teams and make them later. However, they were more than happy to add them to automated testing. So, part of adding 10 new people to automated testing and combining the teams was to get the new developers up to speed, provide resources for automating tests, and reduce reliance on the 8-year-old automated testing framework (i.e., teach veteran automated testers how to use the new automated testing framework MVP).

- **Automated Test Team 1**. Things went remarkably smoothly, as pair programmers used the first Sprint of release two for discovery or Agile Product Management, Lean UX, and story mapping of new automated tests. For the easier tests, an entire two-week Sprint was a bit excessive, but there was still a lot of architectural runway to create for automating the new tests. Once this was done, pair programmers started automating their new tests. The middle Sprints were still a bit rocky, but pair programmers had the process for creating new automated tests down to a science by the end of release two. By the middle of release two, the first three new automated test MVPs were completed, which was a major win for the project using the new automated testing framework MVP. They were deemed MVPs because they would need some more polish in later releases to add hooks for a future reporting capability associated with the new automated testing framework MVP. Some members of team one continued to extend the new automated testing MVP to create new automated tests. So, in actuality, all of team one was focused on creating the three new automated tests. The test engineers were thrilled because the new tests saved hundreds of hours of manual testing, and in some cases, the entire test could actually be run. In other words, the manual tests were so long and complex, they were simply abandoned before they could be completed. All the king's horses and all the king's men simply couldn't run the mountain of manual tests created over the last 10 years. There was a bit of jealousy by the second automated testing team, as the first team had done what they couldn't do for several years. More importantly, pair programmers were often new and even junior developers, while the project had a bias for older multi-decade veterans. In other words, the veterans had little respect for newer developers and often forced them off of the project as quickly as they could. Instead of lending them a helping hand, the second team merely watched the first team's developers make as many mistakes as possible, and simply complained about them at the last minute when there wasn't enough time or capacity to improve the new automated tests. The Scrummaster was thrilled that the automated test teams were making some progress after a release and a half, so some of the criticism of the new automated tests was also aimed at the Scrummaster, because team two didn't want him to succeed either.

- **Automated Test Team 2**. After an entire release—90-day period—Of melting down like an overheated nuclear reactor, the second team kicked into high gear as well. They finally completed their first new automated test due to constant hounding by the Scrummaster. Although Scrummasters are not supposed to be micromanagers, the second team simply had an 8-year-old culture of dragging their feet as slowly as possible. Since their first automated test was nearly complete, the Scrummaster pushed them to demo it to stakeholders as quickly as possible. This isn't to say there wasn't still a lot of work to do on it. However, the first test was assigned to a junior programmer and the senior developer simply didn't want to help him out. At the Scrummaster's urging, the senior developer stepped in to help the junior developer complete the second team's first automated test. It was a good team building experience, because the senior developer needed to be less selfish and bond with the junior developer. At the beginning of release one, the Scrummaster had the junior developer moved next to the senior developer for just this purpose (i.e., bonding). It finally paid off at the beginning of release two (i.e., better late than never). Now that the second team completed their first new automate test, the senior developer could focus on automating the more complex one that project management skewered the Scrummaster over during release two planning. Although, team two planned on using six Sprints to automate their second test, the senior developer automated it in only two Sprints. That is, experience from helping the junior developer automate the first test provided the senior developer with the confidence to automate the complex second new test alone. Of course, these two tests were created using the 8-year-old automated testing framework MVP (not the new one created by team one during release one). At this point, team two bogged down, and although they started creating a third new automated test, they didn't finish it. However, multiple events slowed team two down, including the merger with team one midway through the Sprint, encroachment of the holiday season, and the advent of a major project testing event. That is, team two was made up of senior developers with tons of holiday leave of which they partook in liberal quantities unlike team one.
- **Combined Test Team**. As previously mentioned, the automated test teams were combined midway through release two, 10 new developers were added, and the combined team was now 25 people strong. Again, contrary to the "Mythical Man Month," the combined team's velocity increased rather than slowed. One of the things team one did was streamline the process necessary to onboard new developers from a few weeks and months to a couple of days (or a week at the most). If a new developer didn't onboard within two weeks, then they were simply unmotivated, there was a major problem, or they were incapable of software development. Some of our veteran developers did not get through the onboarding process in under six months and were asked to leave the team. A couple of our newest developers made it through the onboarding process in 48 hours and began contributing new code right away. The old onboarding process was an artifact of the 10-year-old project that was streamlined by team one during release one. This is part of the reason project management added 10 new people to team one and combined the teams. Midway through release two, the test director insisted that the combined team gather up the new product development team tests and run them automatically. That is, incorporate all of the development tests into a DevOps MVP. The Scrummaster suggested that team one's technical lead assume responsibility for this due to his technical prowess. The Scrummaster already knew this was far too difficult for the old team two members. However, the project's chief architect insisted that the combined team create a far more complex DevOps MVP at this point. Both the testing director and the Scrummaster were a bit wary about creating a complex new out-of-scope epic DevOps MVP midway through release two (especially since the holiday season was upon them). Team one's technical lead agreed to take on the challenge and began in earnest. It took a Sprint or two to get the combined team going on the new DevOps MVP. Team one's technical lead formed a small tiger team to begin the DevOps MVP, but they got off to a slow start. The Scrummaster and team two technical lead lit a fire under them because there wasn't much time left in release two. A basic Continuous Integration, Continuous Delivery, and DevOps MVP was quickly started, and the tiger team began focusing upon that. Meanwhile, the Scrummaster onboarded over one third of the combined team, pushed the developers to create six new automated tests, and continue extending the new automated testing framework MVP. It didn't dawn on team one's technical lead how challenging it was to create the new DevOps MVP for the project's second major testing event until the calendar year was about over. At this point, there was only one three-week Sprint remaining in release two and the test director was out of time and patience with everyone. It was the test director's own fault for asking the combined team to create an out-of-scope DevOps MVP at the last minute, and he blamed everyone but himself. Project management had no idea what was happening behind the scenes and continued raining down accolades on the new product development team's Scrummaster for hitting precise EVM cost and schedule targets before the final Sprint, although his tests would not run and could not pass the project's second major testing event. The DevOps MVP tiger team worked 60-80 hours per week over the final three-week Sprint to debug the new product development team's code and tests, extend them to actually run automatically, integrate them into the DevOps MVP, and actually run them in support of the testing event. They did this without the help of the new product development teams which did no constructive work during the final three-week sprint except continue to garner project management accolades and praise. The test director flogged the combined team mercilessly for his own error, stopped talking to everyone, and pretty much gave up hope. The tiger team finally debugged all of the new product development tests, integrated them into the DevOps MVP, and successfully ran them the night before the weeklong final testing event was over. The test director gave the credit for the DevOps MVP to test engineering, not the combined team, and project management gave all of the credit to the new product development team's Scrummaster. By the end of release two, the project delivered a major new product development MVP, automated testing framework MVP, DevOps MVP, eight automated tests, and the automated test teams saved two major testing events with no EVM accolades.
- **New Product Development Teams**. As alluded to earlier, the new product development teams were pressured by the project management team to hit their EVM cost and schedule targets as precisely as possible over the prior six months (releases one

and two). The project management team even highjacked the Scrummaster Community of Practice (CoP) to tattoo the importance of hitting EVM targets on all Scrummasters above all else (halfway through release two). As such, the new product development teams completed their coding with little testing, produced just-enough minimalistic development tests to convince product owners to accept their code at the end of each Sprint, and were unable to run their automated development tests at all in support of major, critical project testing events. This seemed antithetical to the project's major goals of automating every new test from this point forward, creating no new manual tests, and automating the manual tests created over the last decade. That is, either you're going to reward hitting EVM targets and gaming tests or automating tests and gaming EVM targets. As expert after expert have demonstrated over and over again, focusing upon one Key Performance Indicator (KPI) like EVM leads to gaming of other critical variables. This proved true when the new product development tests did not run in the first major project testing event and the test director gamed the testing results, while the new product development team's Scrummaster was heralded as an EVM demigod. Even the automated testing team was susceptible to suboptimization as they were rewarded for creating new automated tests as quickly as possible, but not satisfying all of the automated reporting requirements. Again, focusing on one variable causes gaming of another one. New product development teams gathered test cases, suboptimized them to pass Sprint demos, didn't care if they satisfied the intent of the acceptance tests, and clearly didn't care if their automated tests could not be run over and over again as part of a CI, CD, and DevOps pipeline. That should have been a basic acceptance criterion (i.e., at the end of each Sprint, automated tests are checked into DevOps pipeline, and must pass three successive runs to close the user story). In all fairness to new product development teams, all three architectural runways or MVPs were being created simultaneously (i.e., new product development code, new automated testing framework MVP, and new DevOps MVP). So, it's hard for anyone to hit a moving target, especially if the new product development teams are being rewarded by project managers to fly at light speed toward precise EVM cost and schedule targets. None-the-less, the combined automated test team had to bite-the-bullet, eat the out-of-scope DevOps MVP and new product development technical debt, and successfully run all new product development tests with the DevOps MVP and automated testing framework within a day of ending the project's second major testing event. Of course, all of the politicians rushed in to claim credit for passing the testing event including project management, new product development, and test engineering (but none of it given to the automated testing teams). Much of the responsibility for building the out-of-scope automated testing and DevOps MVPs on a wing and a prayer fell directly on the project management team, not-so agile project management office, test director, and chief architect.

Release Two Scrum Ceremonies

The project's 35 teams continued to use Scrum religiously, and even seemed to hit a rhythm and perform as a team of teams after 10 long years. The project started with 10 or 20 teams, but only a handful of software development teams used Scrum. However, in the last five years, all 20 teams were expected to use Scrum, whether technical, administrative, hardware, or software. Yes, there were still some teams exempt from Scrum like the project management and systems engineering teams which continues to be a big mistake. However, it seems as though Scrum was falling out of favor after the first 10 years of using it, and now there seemed to be a renewed interest in using Scrum among most of the project's teams (and they were getting good at it). There was also an underlying ALM tool that had basic support for Scrum, although it was not consistently used from team-to-team. Everyone did Sprint planning, daily standups, Sprint demos, retrospectives, SoSs, and Release Planning. Yes, there was still some murmuring among the masses, including Agile Coaches and Scrummasters. Scrum is like a runaway freight train that never stops, a little bit like the movie, "Snowpiercer!" Some people affectionately refer to this as the "Tyranny of Scrum" (i.e., Scrum is a very demanding tyrant, sort of like a bad dream that never ends). It takes a few years to get used to the non-stop pace of Scrum's Tyranny, which only becomes completely intolerable when improperly managed (i.e., lack of agile product management, overutilization, and dropping epic MVP technical debt bombs on Scrum teams halfway through a release because you're too cheap to pay for them). Some Scrum approaches like the "Scaled Agile Framework (SAFe)" recognize the importance of allocating resources to epic MVPs such as DevOps architectural runways (but not this project). Scrum and SAFe have to be used properly, and although neither framework is overbearing, humans have the innate ability to use the least number of Scrum and SAFe practices necessary to get the job done. This project's Scrum implementation is a "Feature Factory" sweatshop, which in reality is not all that bad (i.e., after 10 years they've established the basis for doing Scrum correctly instead of pushing out half baked features as quickly as possible). So, the project management team is faced with two opportunities now (i.e., boldly do Scrum correctly or revert to traditional project management practices in order to hit precise EVM cost and schedule targets).

- **Automated Test Team 1**. By the beginning of release two, the first automated testing team had their ceremonies down to a science. Of course, by this time, much of the perturbation associated with establishing the basic automated testing framework MVP was behind them. They could groom a Sprint plan in 30 minutes, do Sprint Planning in 30-minutes, perform standups and thorough Sprint demos, and do quick retrospectives. Something about this team prevented them from having a truly effective retrospective that could really help this team. Sometimes improvements are made behind the scenes rather than in the foreground and the technical lead, Scrummaster, and developers themselves were constantly making necessary course corrections in real time. Sprint demos were a little lengthy for the Scrummaster's taste as most developers were creating working code every Sprint and were eager to show it off. Codifying Sprint Plans in the ALM tool was now pretty routine for the Scrummaster and the technical lead and developers became more adept at recording or logging new out-of-scope work in the ALM Sprint Plan as new work emerged (but not all the time). Some of the new developers were a bit perturbed by the notion that the Sprint Plan was like an iceberg and only about 5% or 10% of the actual work was actually visible in the ALM. The Scrummaster quickly settled them down and everyone became accustomed to using a simple Sprint Plan in the ALM.
- **Automated Test Team 2.** The second automated testing team had their Scrum ceremonies down to a science as well. They could do Sprint planning in about 30-45 minutes, but they followed their Release Plans a bit closer than team one. Their standups were quicker and so were their Sprint demos and retrospectives. The only problem was they felt left behind by team one and were eager to know what was happening beneath the covers of team one. That is, team one was building a new

automated testing framework MVP along with new automated test MVPs, while team two was using the 8-year-old automated testing MVP. Most people believed the older automated testing MVP was now obsolete, so team two felt they were obsolete too. They insisted that teams one and two be combined, and eventually got their wish midway through release two. Now they complained about long Sprint Planning, daily standups, Sprint demos, and even retrospectives to the project management team. First, they went behind the Scrummaster's back to complain about having two automated testing teams, now they were doing the same to complain about combining the two teams. Their best skill was complaining instead of creating new automated tests, which is what they should have been doing. It's important to note that the second team was comprised of senior developers who wanted to micromanage the first team, not compete with the first team's hyper-productive programmers.

- **Combined Test Team**. Although the combined automated testing team now had a whopping 25 developers, this turned out to be a good thing in the Scrummaster's mind. At first, the Scrummaster was worried that the first team's technical lead would quit due to the inordinately large combined team. But he seemed to take it on the chin and welcome the new larger team. That was certainly unexpected. As far as managing Scrum ceremonies, it was easier—less overhead—For the Scrummaster to manage one large team than two smaller teams. There was one Sprint plan to create, one daily standup, one Sprint demo, one retrospective, and one team to track for project SoSs. Furthermore, the combined capacity and load was much higher, the 10 new developers onboarded quickly and began adding value, and there was a lot more working software to demo at the end of each Sprint which made the product owner happier. The Scrummaster didn't have to sugarcoat the performance of the team as much. Everyone had insight into what each developer was doing; there was more synergy, teamwork, and cooperation; and the team could rebound from dips in capacity due to unplanned absences and holidays. More importantly, the combined team was able to absorb the new, unplanned, and out-of-scope DevOps epic MVP a little easier due to excess team capacity. Yes, there was still a free rider or two but now everyone was under pressure to perform at the highest levels as there was nowhere to hide. Another benefit of the combined team was that team one had more coding expertise, but team two had more infrastructure expertise, both of which were necessary for creating new automated tests as well as the new DevOps epic MVP. The velocity of the combined team increased by 3 times to 180 story points—highest in project history (so much for the "Mythical Man Month").
- **New Product Development Teams**. The new product development teams settled down a bit during release two as well. There was a little more cooperation between the four teams than there had been in release one. The new product development teams stopped complaining that the automated testing teams were slowing them down, which was an unfounded claim. Their SoSs were remarkably non-descriptive, but verbally superfluous, which project management encouraged. They finished their code before the final three-week Sprint of release two and declared they would do no productive work during the final 3-week Sprint. This, raised no eyebrows. Meanwhile, the combined automated testing team was working 60-80 hours per week to fix new product development's code, tests, stabilize the new automated testing framework MVP, complete three new automated tests, standup the new DevOps MVP for the project's second major testing event, and run the new product development team's code "successfully!" It is important to note that the Scrummaster of the combined automated testing team socialized with the new product development team's junior developers to understand how they were testing their code. From this, the Scrummaster was able to uncover what they were doing wrong, what systems and test engineering were doing wrong, and how to make critical course corrections during the construction of the DevOps MVP. Basically, the new product development teams were writing their own non-sanctioned development tests to pass their Sprint demos, while the DevOps MVP would run the sanctioned tests. This of course created multiple problems as the formal sanctioned tests needed to be debugged. Team one's technical lead would discover that fact very late, while he admonished the Scrummaster for interfering in technical matters. The other problem, of course, was that neither the sanctioned nor unsanctioned tests were conducive to continuous automation and testing for the DevOps MVP which was also discovered late. Instead of asking new product development to write robust tests, the DevOps tiger team created new tests from the sanctioned and non-sanctioned tests and new automated testing framework MVP.

Analysis and Lessons Learned

There were many striking features associated with this IT project as 80% of it involved operations and maintenance of global data centers and 20% involved creating complex new software. It was a 20-year-old project that had been using Scrum for the last 10 years. For the last five years, about 20 Scrum teams were formed as opposed to about 5 Scrum teams 10 years ago (now there are 35+ Scrum teams on this global data center transformation project). The project management team is wickedly smart, many of them have served as Scrummasters, and quite frankly, they understood Scrum's technical practices inside and out. You're simply not going to challenge the project management team on the accuracy of Scrum practices. However, the project management team failed to make the leap into lean, agile, and innovation "thinking." Models like Scrum are better suited for innovation projects with high uncertainty, a lot of discovery activities, agile product management, Lean UX, and business experiments. Scrum was never intended as an add-on to traditional project integrated master schedules, nor to be used to establish, manage, and hit EVM targets with split second precision. Doing so undermines the very purpose of using Scrum which is to create working software with high business value. Furthermore, Scrum was never intended to create large scale feature factory sweatshops resulting in unstoppable runaway freight trains speeding along at maximum capacity. Yes, agile methods like Scrum were created to focus on creating valuable vertical feature slices with little thought to long term investments in capital intensive architectural platforms. However, Scrum frameworks like SAFe recognize the importance of creating short term, just enough, and just-in-time throwaway architectural runways to support vertical feature slices. In other words, if you're gonna send your kid to the store for a gallon of milk, then at least give him the keys to the car, drive him there, ask the neighbor for a ride, or commission an Uber. Don't just expect your kid to run a mile and back to through the snow five minutes before dinner. Yes, Scrum deemphasizes the role of long-term planning, but never said short term throwaway architectural runway MVPs were out of the question. More importantly, if you're gonna invest in a 10+ year Scrum project, then keep on moving forward rather than backsliding into traditional project management and precision EVM cost and schedule targets. And, of course, invest in Agile Product Management (APM), Lean UX, and Business Experiments, don't just expect development teams to take architectural runways out-of-hide. Respect Scrum if you're going to use it, don't just give Scrum lip service, and constantly drop epic MVPs on Scrum teams after release planning.

- Hire, train, and/or certify a lean, agile, and innovation project management team.
- Establish an Agile Product Management (APM), Lean UX, and Architectural Runway team.
- Reduce the load of all Scrum teams and coach them to perform agile product management.
- Train, enable, and coach Scrum teams to form rational quarterly release plans well in advance.
- Reward teams for achieving Scrum values vs. traditional project management values like EVM targets.
- Respect the Scrum release and sprint planning process and don't subvert it for personal aggrandizement.
- Don't treat Scrum teams like your personal services teams for frequent unplanned out-of-scope epic MVPs.
- If you need architectural runways, then resource these teams and build them in advance like normal managers.
- If you want a DevOps pipeline then plan one, fund it, and build it in advance (don't expect it to magically appear).
- Consider 3-week Sprints to ease Scrum's tyranny, standardize shorter story points, and reward hitting Scrum goals.
- Treating people well goes a long way, so don't treat people like dispensable cogs that cost millions of dollars to replace.
- If you want to achieve a sustainable pace, then don't use Scrum as a runaway freight train (succumb to tyranny of Scrum).
- Don't succumb to myth that Scrum can fix a brittle antiquated information technology stack with overwhelming technical debt.
- Don't use Scrum in high power-distance corporate cultures where project managers make centralized decisions for workerbees.

Summary

In mid to late 2021, a large IT project chartered two automated test Scrum teams to create as many new automated tests as possible, and two new Scrum product development teams to create as much new software as possible with precise EVM targets. The project management team had been using Scrum as a sweat shop feature factory approach with full utilization, large batches, and unstoppable ceremonies. In fact, this was a hybrid traditional-agile project, in which project managers and other senior leaders frequently used Scrum teams for personal services, dropping unplanned, out-of-scope epic MVPs on them in between release and sprint planning events. Project managers didn't even stop long enough to charter Agile Product Management (APM), Lean UX, Discovery, business experiment, nor platform teams to create the architectural runways sorely needed by feature teams. In some sense, they expected teams to self-organize, form their own governance, perform their own discovery, and build their own architectural runways. While this doesn't sound too bad, some of these architectural runways took months or years to create, constituting unplanned and under-resourced epic MVPs, features, and user stories. One such architectural runway was a establishing a DevOps pipeline, which was a major project goal, but was not directly funded, planned, nor resourced. Of course, all architectural runways and new product development epics and features had to be created on a highly-brittle, highly unstable, and extremely antiquated technology stack (hardware platform), which lost the race with Moore's Law decades ago. One small ragtag group of software developers had the courage to plan the development of a new automated testing framework MVP before creating new automated tests. Both of course were the basis for an application-level DevOps pipeline, but not an infrastructure DevOps pipeline. Then the test director dropped the bomb, chartered an unplanned, out-of-scope DevOps epic MVP on the automated testing team, which they ate and created in about two months working 60-80 hours per week throughout the holidays while the senior staff and developers took extended holiday leave. Meanwhile, new product development teams were constantly rewarded for hitting precise EVM cost and schedule targets with split second accuracy although their code did not work and neither did their tests. As usual, some teams will rise from the flames like a phoenix and win the day, at least temporarily, but not without a very high personal cost. All of this nonsense is completely avoidable with proper leadership, agile project management, agile product management, lean thinking, lean UX, and a host of agile planning values, principles, practices, tools, and measures.

Further Resources

- Coupland, M. (2021). *DevOps adoption strategies*: *Principles, processes, tools, and trends*. Birmingham, UK: Packt Publishing.
- Heath, F. (2020). *Managing requirements the agile way*: *Bridge the gap between requirements and executable specs*. Birmingham, UK: Packt Publishing.
- Hofer, S., & Schwentner, H. (2022). *Domain storytelling*: *A collaborative, visual, and agile way to build domain-driven software*. Boston, MA: Pearson.
- Levitt, S. D., & Dubner, S. J. (2006) *Freakonomics*: *A rogue economist explores the hidden side of everything*. New York, NY: HarpersCollins.
- Levitt, S. D., & Dubner, S. J. (2009) *Super freakonomics*: *Global cooling, patriotic prostitutes, and why suicide bombers should buy life insurance*. New York, NY: HarpersCollins.
- Parker, M. (2021). *Humble pi*: *When math goes wrong in the real world*. London, UK: Riverhead.
- Pearson, C., & Porath, C. (2009). *The cost of bad behavior*: *How incivility is damaging your business and what to do about it*. New York, NY: Penguin.
- Rico, D. F. (2017). Business value, roi, and cost of quality (coq) for devops. http://davidfrico.com/rico-devops-roi.pdf
- Rico, D. F. (2019). *32 attributes of successful CI, CD, and devops.* http://davidfrico.com/devops-principles.pdf
- Rico, D. F. (2021). *32 attributes of successful U.S. DoD cloud computing.* http://davidfrico.com/dod-cloud-principles.pdf
- Rico, D. F. (2021). *33 principles for improving SAFe developer experience* (*DX*). http://davidfrico.com/safe-dx-principles.pdf
- Rico, D. F. (2021). Business value of CI, CD, & DevOps. [Brief] http://bit.ly/34vHOcl [Video] http://bit.ly/3HnluQJ
- Rigby, D., Elk, S., & Berez, S. (2020). *Doing agile right*: *Transformation without chaos*. Boston, MA: HBS Press.
- Rupp, C. G. (2020). *Scaling scrum across modern enterprises*. Birmingham, UK: Packt Publishing.
- Rupp, C. G. (2021). *Driving devops with value stream management*. Birmingham, UK: Packt Publishing.
- Savage, S. L. (2012). *The flaw of averages*: *Why we underestimate risk in the face of uncertainty*. Hoboken, NJ: John Wiley.
- Smil, V. (2021). *Numbers don't lie*: *71 stories to help us understand the modern world*. New York, NY: London.
- Varma, T. (2015). *Agile product development*: *How to design innovative products and customer value*. New York, NY: Apress.
- Viki, T., Strong, C., & Kresojevic, S. (2021). *Lean product lifecycle*: *A playbook for products people want*. Harlow, UK: Pearson.

Observations, Emerging Challenges, Patterns, and Highlights—From Releases Three and Four

Although a basic DevOps MVP was in place by the end of Release Two, it took another release to stabilize it. It took another 90 days for six or seven top notch developers to refine the underlying DevOps pipeline along with any automated tests riding on top of it produced by new product development teams. Although all three elements—basic DevOps pipeline, new product development MVPs, and automated tests ran at least one time after the end of Release Two, there was still a lot of work to do. The underlying DevOps pipeline had to be extended, made more robust, and more resilient to failure. And, of course, the kinks had to be worked out of the complicated automated tests as well. The exacerbating element was the instability of the underlying hardware and software. System outages and downtime were all-too common hindering the ability to get consistent runs of the DevOps pipeline and automated tests. The primary DevOps team just debugged one DevOps pipeline function and one automated test at time over a period of 90 days until the DevOps jobs ran for four or five days in a row without failure. As exciting as this was, there were still dozens of developers refactoring the client user interface that could break the DevOps pipeline and automated tests at any time. In other words, there was far too much development churn for the current DevOps architecture to manage.

A major goal for the next 16 releases was to implement DevOps in order to shift left, reduce the overhead on late manual testing, and increase the deployment speed so end users could get value quicker. However, there were many challenges to this basic vision, one of which is that leadership didn't plan on any direct investments in an expensive DevOps pipeline with a long lead time (i.e., they assumed it would just emerge). Part of that basic assumption was that automated testing teams would create the DevOps pipeline to shorten lead times of testing, so that was the extent of shifting left leadership imagined. They weren't DevOps experts, and they had no idea that shifting left meant new product development teams would absorb the investments in frequent automated testing and deployment. In other words, leadership assumed new product development teams could continue throwing code over the wall to testing teams as they had for the last decade (and testing would shift left into automated vs. manual testing). Worse yet, solution and product managers kept new product development MVPs small and tight, built in plenty of extra capacity into their schedules, and graded them on how fast they completed. No one ever established any expectations that new product development teams were responsible for building and testing production (deployable) code.

Much of the philosophy and culture surrounding the last 10 years of this project was due to the fact that the chief Scrummaster was a former developer who favored a laissez faire approach to new product development. Furthermore, the chief Scrummaster didn't understand DevOps and was not part of the DevOps vision team. However, the Chief Scrummaster retired early in Release Three, the chief DevOps visionary began self-actualizing a bit, and leadership ever so slowly started to come around and realize shifting left wasn't for manual testers only. However, this new leadership realization didn't start taking place until the end of Release Three and beginning of Release Four. By then it was a bit too late as new product development teams had three releases under their belts applying the throw-half-baked-MVPs-over-the-wall-approach for manual and automated testing teams to harden. Although new product development teams were asked to shift left and assume the responsibility for building production code, automating their own tests, and running them until they pass, these concepts were simply too much. New product development teams were moving so fast building and being rewarded for producing half-baked MVPs, they simply didn't realize the DevOps team spent three releases shifting left without them (while they shifted right).

The DevOps team was doing so well shifting left that solution and product management thought it was natural for new product development teams to keep throwing half-baked MVPs over the wall for DevOps to harden. The DevOps team suffered silently working overtime for three releases, while new product development teams worked at less than 50% capacity. The DevOps Scrummaster had some experience as a cost estimator, created a model illustrating the effort required to standup the DevOps pipeline, and helped illuminate the basic problem. That is, that shifting only half a step to the left was economically unfeasible, especially with the single largest program or project testing event looming on the horizon. In this case, people seemed to begin waking up to the problem with the presence of hard facts, figures, and statistics illustrating the extent of the misunderstanding. W. Edwards Deming said, "In God we trust, all others must bring data," and that certainly proved true in this case. However, basic beliefs are hard to overturn with data as illustrated by the old adage, "there are lies, damn lies, and then there are statistics!" It's safe to say that the most persuasive element turning the ship around was not the data, but the departure of the chief Scrummaster and self-actualization of the DevOps visionary with whom leadership intuitively trusted (without data).

The larger challenge is going to be the execution of major new test events, system releases and deployments, and scaling the number of new product development teams and the robustness of the DevOps pipeline. Many first time DevOps projects make the mistake of first standing up simple DevOps platforms and then investing in small to medium-sized dedicated DevOps data centers. However, then long lead item brick-n-mortar DevOps pipelines become the resource bottleneck. The better solution is always to implement virtually scalable DevOps pipelines using commercial cloud services that can be instantly terraformed—Stood up and torn down—Without investments in long lead item, resource intensive, dedicated brick-n-mortar (physical) DevOps assets. DevOps initiatives are not different from any other transformation and must go through evolutionary stages and can rarely leapfrog to the end game.

Rico, D. F. (2017). Business Value, ROI, and Cost of Quality for DevOps. http://davidfrico.com/rico-devops-roi.pdf
Rico, D. F. (2019). 32 Principles and Practices of CI, CD, and DevOps. http://davidfrico.com/devops-principles.pdf
Rico, D. F. (2021). Business value of CI, CD, & DevOps. [Brief] http://bit.ly/34vHOcl [Video] http://bit.ly/3HnluQJ

Observations, Emerging Challenges, Patterns, and Highlights—From Releases Four and Five

The prior Release Three was spent stabilizing the new DevOps MVP which was not an easy feat. In Release Two, the DevOps team not only had to create an operational DevOps pipeline but put it to the test with a major system release. Therefore, Release Three was a 90-day period used to stabilize the rough ride experienced during Release Two. Release Four would prove to have its own set of unique challenges. Since Release Three was spent stabilizing the new DevOps pipeline, Release Four was spent hardening the DevOps pipeline even more, expanding the number of tests handled by the DevOps pipeline, and improving the UX and look and feel of the DevOps console, reports, and output. Release Five consisted of refactoring, streamlining, and expanding the capabilities of the DevOps pipeline, preparing it to handle more MVPs, five concurrently to be exact, and actually supporting a second major system release. The refactoring was no small feat and the DevOps product owner and scrummaster felt a bit uncomfortable about the amount of change to which the DevOps pipeline was subjected so close a major system release. Furthermore, the principal DevOps designer left the team just a few sprints before the second major system release. To add insult to injury, the four new MVPs would not be ready until the last sprint of the quarterly reporting period in big bang style.

The operational environment was established fairly late in the quarter, while the DevOps pipeline itself continued to undergo major reconstructive surgery. The test manager also retired by the end of Release Four, and the new test manager did not get proactively involved until the last sprint of the quarterly reporting period as well. The DevOps scrummaster had to get the party started and marshal the MVP developers to start preparing for major system release two (although they hadn't even started coding two of the four major MVPs). Special testing servers were also requested by two of the MVP teams, so the scrummaster also had to take charge of setting up new DevOps servers. The first two new MVPs started running on the DevOps pipeline, while the DevOps developers were still refactoring the DevOps pipeline itself, the last two MVP teams started coding, and the scrummaster was standing up the new DevOps servers. Literally days before the scheduled start of major system release number two, the final two MVP teams completed their code. This was a last-minute rush job and the MVPs had to be radically downsized to be completed in the last sprint of the quarterly reporting period. The new DevOps server was finally complete and the final two MVP jobs were fed to the DevOps pipeline and everything was looking pretty good up until this point.

The DevOps team and four MVP teams ran their DevOps jobs automatically, passed, and completed the test event within 48 hours of the actual testing event itself. The DevOps team continued to finish their refactoring of the DevOps pipeline, resolve functional and performance deficiencies, and ensure the four new major MVPs executed as smoothly as possible. Everyone was surprised how well things were moving along. There was a bit of program wide resentment towards the DevOps team for the speed at which the four new major MVPs were tested. The new test manager added new requirements to the DevOps teams on how to deliver their test results to external stakeholders, customers, and end users which was a bit annoying. New methods of performing customer demonstrations were devised consisting of technical briefings, live demos, and DevOps reporting. The real credit went to the technical leads of the first DevOps team for establishing the testing platform(s), the second DevOps team for stabilizing, refactoring, and streamlining the operation of the DevOps pipeline, and of course, the MPV developers themselves for writing solid automated acceptance tests to feed the DevOps pipeline. The results were almost too good to be true at this point, because the scrummaster's worst nightmare was that the DevOps pipeline wouldn't scale beyond one MVP.

The other shoe finally dropped, two of the four MVPs had been streamlined too much, and they decided to add more features to their codebase in the final sprint. Unfortunately, this took an entire month, i.e., it bled into the next quarterly release and the second major testing event was now behind schedule. The real cause of the delay was that two of the four MVP teams didn't start coding until the 11th of 12 sprints consisting of two quarters, otherwise they could and should have finished their code much earlier. Its unclear why these two MVP teams fell into the trap of developing 11 sprint big bang releases. Perhaps, there was simply too much complexity, uncertainty, and learning to be done before the MVP teams were confident enough to finish their code. Perhaps the scope of the MVPs was simply too large. Perhaps, they simply didn't understand the DevOps mindset of coding and testing a little bit at a time each sprint and feeding small chunks to the DevOps pipeline a little bit at a time. And, of course, why did the DevOps team decide to undergo a major system refactoring itself leading up to the second major system release. One thing is for sure, the DevOps culture is still young, so is the DevOps pipeline, it hasn't been scaled out completely, and there are many more MVPs on the horizon including stringent requirements for future system releases on the production fabric.

Much work has yet to be done. The DevOps culture must continue to mature. The DevOps pipeline must also be expanded and improved. More importantly, MVP teams should be making DevOps drops multiple times each sprint instead of waiting for bi-annual program testing events. The first MVP may have to be refactored a bit, as it is the most unstable of the five program MVPs to-date. It's unclear whether the instability is in the system code itself, IT infrastructure, DevOps pipeline, or automated acceptance tests. Needless to say, the program's first five MVPs must run in the DevOps pipeline forever, they must run routinely, and they should succeed every day. While the DevOps team will spend some time stabilizing the five major MVPs, this will continue to be a very tall order for everyone. The IT infrastructure itself must continue to bake, mature, and be adapted to the brave new world of DevOps computing.

Rico, D. F. (2017). Business Value, ROI, and Cost of Quality for DevOps. http://davidfrico.com/rico-devops-roi.pdf
Rico, D. F. (2019). 32 Principles and Practices of CI, CD, and DevOps. http://davidfrico.com/devops-principles.pdf
Rico, D. F. (2021). Business value of CI, CD, & DevOps. [Brief] http://bit.ly/34vHOcl [Video] http://bit.ly/3HnluQJ

Challenges of Dynamic Demand in U.S. Public Sector Acquisitions

Problems/Challenges

- PM lacks empathy/overreacts
- Uneven demand/utilization
- Burnout/frustration/attrition
- Uneven workload distribution
- Indecisive/demanding customers
- Constant customer turnover
- Uneven resource consumption
- Multitasking/poor quality/lateness

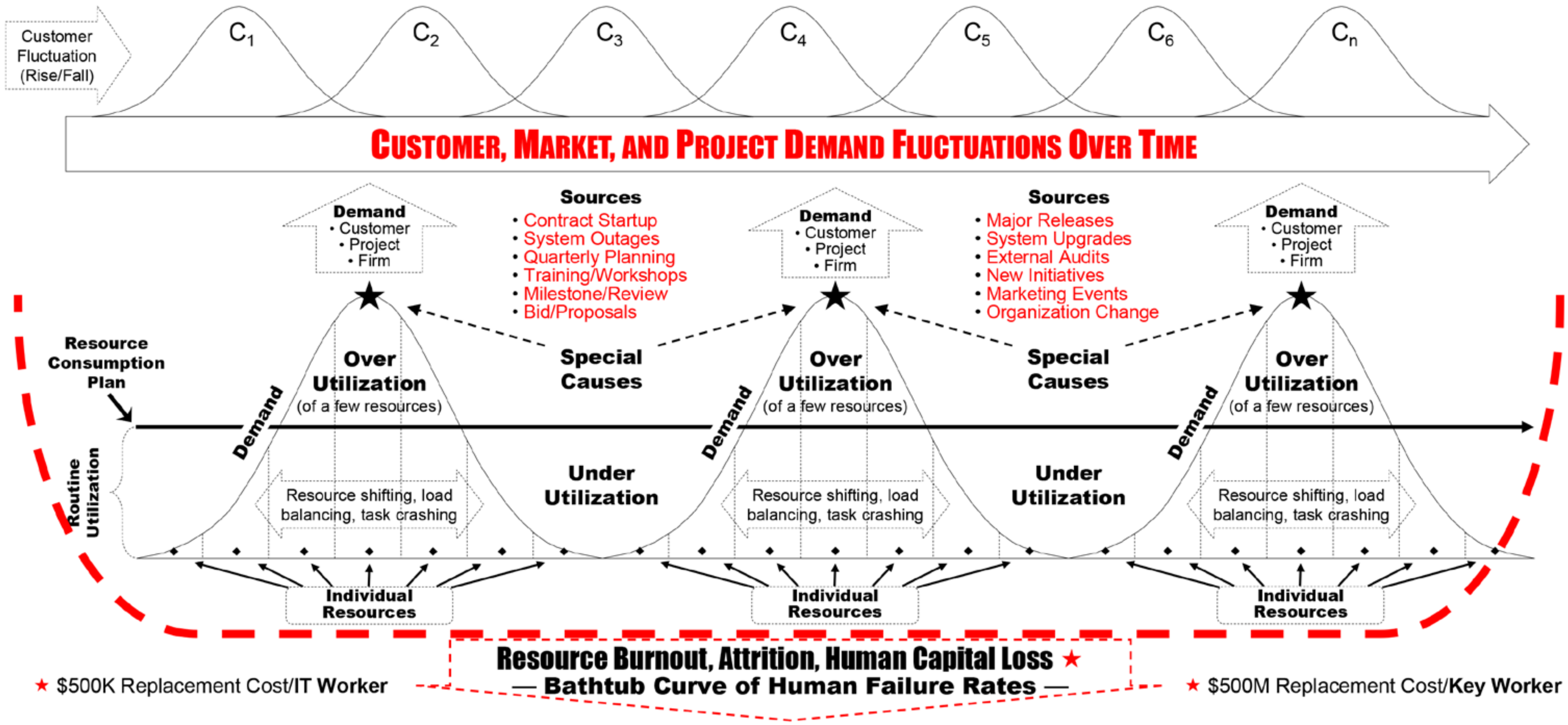
Acquisition Goals

- Produce quality products
- Teamwork/collaborate/communicate
- Match capacity to demand
- Sustain investment in human capital
- Level the workflow/sustainability
- Don't overreact to special causes
- Create positive/safe environment
- Satisfy customers/be more successful

Solutions

- Small teams matched to MVPs
- Lean-Kanban with low WIP limits
- Visualize/simple/communicate
- Collaborate/trust/camaraderie
- Extreme scope/design simplicity
- Modularizations/few dependencies
- Build-in extra margin/capacity
- Lean-Kanban intake process

- Government customers are happy contractors are spending their money (and they're getting mission value)
- Contractors happy to spend money, get job done, and reasonably meet government customer expectations
- Corporations are happy if they get revenue, win newer contracts, and manage attrition (burnout) effectively



Customer Fluctuation (Rise/Fall)
C_1
C_2
C_3
C_4
C_5
C_6
C_n
CUSTOMER, MARKET, AND PROJECT DEMAND FLUCTUATIONS OVER TIME
Demand • Customer • Project • Firm
Sources
• Contract Startup
• System Outages
• Quarterly Planning
• Training/Workshops
• Milestone/Review
• Bid/Proposals
Sources
• Major Releases
• System Upgrades
• External Audits
• New Initiatives
• Marketing Events
• Organization Change
Special Causes
Resource Consumption Plan
Demand
Over Utilization (of a few resources)
Under Utilization
Routine Utilization
Resource shifting, load balancing, task crashing
Individual Resources
Resource Burnout, Attrition, Human Capital Loss ★
— Bathtub Curve of Human Failure Rates —
★ $500K Replacement Cost/IT Worker
★ $500M Replacement Cost/Key Worker