The Pros and Cons of Agile AI Technology for Neurodivergents

Abstract

Bottomline Upfront. *Modern information technologies are a boon for today's enterprises which are comprised of employees with broad neurodivergent capabilities ranging from extremely fast, wide, and deep cognitive speeds and abilities (vs. those who are slower, narrower, and shallower). That is, they help mitigate and mediate the cognitive distance between neurotypes who are superior at synchronous communications and neurodivergents who are stronger at asynchronous communications. We've reached a tipping point where the capabilities of neurodivergents who represent the largest population of employees may be able to keep pace with neurotypes (if not surpass them).*

Motivation. *Today's enterprises must operate at previously unprecedented speeds and paces to thwart or disrupt global competition with innovatively new products and services ahead of market challengers. Applying lean and agile frameworks like the Scaled Agile Framework (SAFe), Scrum, Scrumban, DevOps, Business Experiments, and Cloud Computing enable top firms to introduce new innovations with the shortest lead and cycle times. This places great cognitive demands on enterprise workforces (employees) for greater speed, capabilities, flexibility, talent, skills, resilience, perseverance, and cost effectiveness (who often represent less than 5% of the total workforce).*

Problem. *The main challenge is that the global talent pool consists of a widely ranging neurodivergent workforce with differing speed, capabilities, and communication quality. Human communications are part and parcel to lean and agile frameworks, as humans must cooperate to identify market needs, solutions, implementations, performance, and rapid continuous improvements to converge on a solution in seconds, minutes, or hours (or discard it completely). A neurotype or neurodivergent with an Attention-Deficit/Hyperactivity Disorder (ADHD) may verbally communicate at 100 mph (with or without making any sense at all). While a neurodivergent with cognitive and verbal delays may require a few minutes, hours, or days for new information to matriculate across their neural networks before composing and (a possibly more rational) asynchronous response. So, this places unprecedented pressure on how widely ranging neurodivergent lean and agile teams can function with optimal efficiency and effectiveness (if at all).*

Approach. *Our approach is to examine the attributes, characteristics, skills, and capabilities of neurotypes vs. neurodivergents with respect to their communication styles. We will also compare and contrast the disparities, differences, and uniquenesses of each communication style with or without the aid of both synchronous and asynchronous information technologies. We will illustrate how the advent of the electronic computer implicitly improves the ability of a widely ranging neurodivergent workforce to communicate, how modern information technologies close this gap even further, and how future technologies may or may not dissolve these differences. More importantly, we'll illustrate how to leverage current and emerging information technologies to continue closing communication gaps leading to even greater global enterprise performance.*

Results. *Our results are to design, develop, and recommend one or more frameworks for classifying and typing the communication abilities of neurotypical and neurodivergent professionals, how information technologies have supported neurodivergents since the advent of the electronic computer, and how modern and emerging information technologies have all but closed this gap. In doing so, we present a broad array of frameworks, tools, and methods to help modern enterprises leverage widely ranging neurodivergent workforces for continued global success.*

Conclusion. *Our conclusion is that modern enterprises are faced with previously unprecedented challenges when it comes to competing on speed and widely ranging neurodivergent workforces and their communication speed and abilities that challenge this goal, however, modern information technologies have come a long way towards closing these communication gaps and continue to do so. It's no longer a matter of either or (i.e., we have a neurotypical workforce or a neurodivergent workforce, but not both). Rather, modern and emerging information technologies help today's enterprises achieve all goals simultaneously (i.e., compete on speed, innovation, quality, and cost; leverage a widely neurodivergent workforce; and utilize modern lean and agile frameworks to achieve short lead and cycle times).*

Footnote. *We artificially categorize and describe (normal) neurotypes as fast synchronous communicators, we lump ADD or ADHD neurodivergents into this fast communication category, and we artificially categorize and describe neurodivergents as slower asynchronous communicators (due to common cognitive and verbal delays). However, there is another category of neurodivergent which is often conflated with neurotypes (i.e., a frontal lobe dominant neurodivergent who can quickly formulate a narrow one-sided logical conclusion without empathy, sympathy, or emotion which may not consider all of the factors). For instance, this neurodivergent may say, the fastest route from point A to point B is by steamrolling everyone in between both points (which may seem like a logical conclusion to them since they have a very weak or holistic theory of mind, emotional intelligence, and poor neurological wiring).*

Introduction

At this point we usually begin by describing the sudden Pre-Cambrian explosion of lean and agile frameworks, their immense popularity and applicability to the globally fierce marketplace, and why they not only help the fittest, but precipitate survival of the fittest as well. However, let's take a giant step backwards and examine the emergence of traditional linear waterfall systems and software frameworks, lifecycles, and methodologies. That is, when, how, and why they emerged and the technological and market forces that precipitated their creation, popularity, and promulgation. This is really the place where neurodivergence first reared its ugly head in the primordial soup, birth, and childhood of electronic computers, operating systems, and application software. Obviously, electronic computers emerged after decades of trial and error in the 1940s to help the U.S. military replace human mathematicians or "computers" with an automated means of calculating ballistics trajectories. Unfortunately, World War II came to a sudden end before they could be properly deployed. The 1950s saw the rise of custom "one-off" military computer systems for controlling and launching intercontinental ballistic missiles. And the 1960s saw the rise of mass-produced electronic computers for accounting and billing purposes among commercial enterprises (i.e., payroll, taxes, banking, general record keeping, financial reporting, etc.).

A smorgasbord of mainframes emerged in the 1960s from the likes of IBM, Burroughs, UNIVAC, NCR, Control Data Corporation (CDC), Honeywell, and General Electric (GE). However, the most popular mainframe in the 1960s was the IBM System/360 which took nearly a decade, 1,000+ people, and $5 billion to develop. In today's dollars, that's about $55 billion. The retail price was about $2 million, which is over $20 million in today's dollars. And they ranged in size from 2 tons to over 15 tons (or 30,000 pounds). When we add in power, space, cooling, and personnel, the big ones certainly lived up to their reputation as "building-sized" computer systems. Thomas Watson estimated that only five mainframes needed to be built to satisfy the globe's demands for business computer systems or mainframes. In the end, over 12,000 IBM 360 mainframes were sold.

Several notable events happened about this time:

(1) General purpose programming languages emerged such as FORTRAN and COBOL.

(2) There were only a few thousand U.S. programmers.

(3) Software applications were custom-made.

(4) Computer programming was learned on-the-job.

(5) There were few Computer Science graduates (although programs existed for a decade).

(6) Most IBM competitors went out of business.

(7) IBM did not allow firms to develop applications for their mainframes.

Basically, IBM held a monopoly on mainframes, operating systems, and early application software (of which there was little to none). IBM initiated several other innovations about this time as well: (1) They applied linear manufacturing waterfall lifecycles and processes to computer programming projects, (2) They applied manufacturing schedules to computer programming, (3) They applied hardware quality control to computer programming, (4) They developed catalogues and libraries of documentation to accompany computer programs, (5) They applied statistical process control (SPC) to computer programs, (6) They applied rigid governance boards and committees to oversee computer programming, and (7) They developed some of the first metrics, models, and measurements to computer programs and projects. Some notable developments from IBM also included "Structured Design," "Chief Programmer Teams," "Software Inspections," and the application of "Manufacturing Maturity Models" to computer programming (i.e., measuring human performance like manufacturing machines). Several other developments emerged about this time like the term "Software Crisis" to denote the unmet demand for computer programs and "Software Engineering" and "Software Factories" as a means of solving the "Software Crisis." **Note**. It's important to mention that the term "Management Crisis" emerged in the late 1950s to describe the complexities of building custom, one-off computer systems before the advent of commercially mass-produced mainframe computers.

The U.S. DoD spent over $1 billion institutionalizing IBM's manufacturing maturity models to slow DoD acquisitions down to a snail's pace (and manage humans like machines) ...

IBM software executives and managers, with their plethora of manufacturing processes and measurements, immediately detected the productivity between the best and worst computer programmers. Even Frederick Taylor could and did do similar measurements with a stopwatch in the late 1800s. That is, there was a 10,000 to 1 difference between the best and worst mainframe computer programmer. Part of the manufacturing discipline (i.e., governance, schedules, waterfall lifecycles, documentation, quality control, metrics, etc.) was to slow the process down, record all tacit (unwritten human knowledge or expertise) in the form of explicit documentation for all to share, see, and measure. In other words, take that expertise out of the heads of the best computer programmers and put it in writing for all of the worst computer programmers to learn and master. This included schedules (timelines), system requirements, architecture, designs, pseudocode, code comments, user documentation, tests, metrics, quality control documents, governance charters and decisions, and manifests of the final products. Basically, IBM discovered that there was a wide range of employees with differing cognitive capabilities, speeds, abilities, and distance (i.e., they discovered neurodivergence). And the answer was to level the playing field (i.e., all programmers would move down the playing field together at the same pace).

Today's Smartphones are infinitely more powerful than the IBM 360, they weigh far less than 30 tons, and they fit in your pocket too ...

IBM had cornered the market, held a monopoly on computers and software, sued its competitors back to the Stone Age, and there was no global competition for application software. In fact, the commercial software industry wouldn't be born until 1970 when the U.S. Department of Justice (DoJ) threatened to divide IBM into a hardware and software firm. There was no such thing as competing on speed, lead and cycle times, innovative products and services, or beating out global competition in seconds, minutes, and hours. It took billions of dollars, thousands of people, and decades to make simple software releases that one person can do today with a single keystroke in seconds using the Internet and Cloud Computing technologies. IBM's answer to the "Software Crisis" was to slow down vs. speed up, share all knowledge by filling libraries full of documentation, and form rigid governance boards to punish anyone making unsanctioned software changes without explicit committee approval in triplicate. But, in spite of this immense software bureaucracy meant to slow the pace of innovation down to the glacial speed of 1960s computer systems, small rogue teams of highly talented developers using fast synchronized human communications formed beneath the radar of punishing governance boards to get the job done in the best way they knew how. That is, the tiny seed of hope in a vast sea of bureaucratic darkness had been planted for a brighter day of lean and agile thinking (along with a breach in its neurodivergent Band-Aid).

Modern Age

Too much transpired between the dark days of IBM 360's reign of tyranny and the modern age to be recorded here. For instance, the U.S. DoJ enabled commercial firms to commercialize application software for IBM's mainframes. A plethora of startups formed in the 1970s from large semiconductor and mainframe manufacturers to create smaller, faster, innovative, and inexpensive microprocessors, computers, operating systems, and applications. The Unix operating system and C programming language emerged created by two programmers which still lives on today. Dozens of microcomputer kits and complete products emerged which are too numerous to mention here. However, the rockstars from this era were the Apple I and II, RadioShack TRS-80, and Commodore PET (which later became the Commodore 64). Apple's Macintosh and its operating system along with Microsoft Windows and Microsoft Office emerged. The Internet and World Wide Web (WWW) emerged (along with hundreds of millions of worldwide websites). Of course, the worldwide population is 8 billion people, 5 billion have smartphones, and 6 billion use the Internet every day to communicate. From a manufacturing perspective, Western firms coined the term "Lean Thinking" to exemplify Japan's unique capability to compete or thwart global competition based on the shortest lead and cycle times. From this Armageddon of glowing, red hot innovations, lean and agile frameworks, business agility, and fast two-pizza teams synchronously communicating in real-time came into sharp focus. The time was now to discard the bureaucracy IBM created in the 1960s to slow communications to a glacial speed.

Hundreds of software frameworks emerged from 1970 to 2000 to balance the perceive need for explicit and tacit communications. That is, just enough processes, documentation, bureaucracy, and creative freedom to get the job done. We fail to mention that the overwhelming majority of software products were created by one person firms with literally no bureaucracy at all. They simply hired the best and brightest computer programmers (i.e., those with a 10,000 to 1 productivity capability), handed over their source code (i.e., all the documentation they had or needed), and asked them to fix bugs or add new features in 90 day increments with little more than a handshake, non-disclosure agreement (NDA), or other short person-to-person contract. If you have that much talent, you really don't need too many synchronous real-time communications. A few grunts, burps, and scratches will suffice. Scrum and Extreme Programming (XP) emerged by the mid-1990s, and the Agile Manifesto was codified in 2001, which read:

> ***We are uncovering better ways of developing software by doing it and helping others do it.***
>
> ***Through this work we have come to value:***
>
> - ***Individuals and interactions*** *over processes and tools.*
> - ***Working software*** *over comprehensive documentation.*
> - ***Customer collaboration*** *over contract negotiation.*
> - ***Responding to change*** *over following a plan.*
>
> ***That is, while there is value in the items on the right, we value the items on the left more.***

That's it, that's the whole ball of wax. People and communications matter more than processes and tools. A working software application is more important than a library of documentation. Talking to end-users and customers is superior to a lengthy legal contract. Frequent change, continuous improvement, and new innovations are more important than project plans, schedules, and business or systems requirements documents. Notice what's not here (i.e., lifecycles, processes, gate reviews, documents, metrics, tools, governance boards, etc.). Let's take a closer look at two of the earliest agile frameworks, XP and Scrum:

Agile Framework	Major Practices
Scrum	• **Sprint Planning** - Lightweight 14–30-day plan. • **Timeboxed Sprint** - 14–30-day planning period. • **Daily Scrum** - Daily 15-minute team meeting. • **Sprint Review** - Code demo at end of planning period. • **Sprint Retrospective** - Short process improvement activity. • **Backlog Refinement** - Prepare backlog for next Sprint.
Extreme Programming (XP)	• **Planning Game** - Lightweight 90-day and 14-day plans. • **Small Releases** - Lightweight 90-day code releases. • **Metaphor** - Shared mental model, metaphor, analogy, etc. • **Simple Design** - Least complex short-term design. • **Tests** - Unit, behavioral, and acceptance tests. • **Refactoring** - Continuous code simplification. • **Pair Programming** - Collaborative development. • **Continuous Integration** - Automated build and testing. • **Collective Ownership** - Public code ownership and changes. • **On-Site Customer** - End-user or customer part of team. • **40-Hour Weeks** - Sustainable pace, no overtime, limited WIP. • **Open Workspace** - Synchronous and osmotic communications. • **Flexible Practices** - Improve technical practices.

Emergence of Asynchronous Technologies

The notion that asynchronous technologies for neurodivergents is something new, emerging, or future is a bit of a misnomer. Asynchronous technologies have been subtly emerging for the last 70 years since the advent of the electronic computer itself circa 1945. The electronic computer was designed to replace or assist human computers or mathematicians in performing live real-time ballistics calculations during actual battles. The best humans, like the best programmers, were thousands of times faster than the slowest mathematicians. But the notion that an electronic computer could replace all humans, consistently perform calculations (even if slower or possibly erroneously), and do so largely asynchronously was a major game changer in terms of neurodivergence. Worldwide firms were no longer dependent upon the best and brightest human computers, a machine could perform calculations for enterprises, and humans could operate them without regard to their position on the neurodivergent spectrum. In fact, cognitively slower neurodivergents may have had an advantage in that programming them was a very slow, methodical, and mentally driven process (i.e., just the right cognitive speed for slower neurodivergents).

Phase	Description
0	Electronic Computers and High-Level Computer Programming Languages.
1	Linear Waterfall Process, Document, and Governance-Driven Systems Lifecycles.
2	Software Factories to Automate System Lifecycles, Documentation, and Testing.
3	Structured and Object Oriented-Driven Computer Aided Software Engineering (CASE).
4	Early Agile Frameworks like Participatory Design, Scrum, and Extreme Programming (XP).
5	Early Web Technologies like Bulletin Boards, Email, Blogs, Chat Rooms, and Messaging Apps.
6	Early Agile Application Lifecycle Management (ALM) Tools like Jira, Gitlab, TFS, Jazz, Confluence, etc.
7	Early AI-Driven ALM Tools like Asana, Ayanza, Clickup, Wrike, Height, Hive, Butler, Copilot, Jira ML, etc.

As we explained earlier, Generation 1 asynchronous technologies were specifically designed to level the playing field between neurotypes and neurodivergents. This was done by slowing the systems development process down to a snail's pace, documenting all tacit knowledge, forming governance boards and committees to make all decisions by consensus, ensuring every process step was documented and enforced, and doing away with rogue neurotypes that could complete complex computer programs in a few days or weeks by themselves. Generation 2 asynchronous technologies were but a mere extension of systems lifecycles as Conway's Law was exploited to automate the system lifecycles in step-by-step fashion just as they appeared in writing with little variation. The advantage to this approach was that computers would assist humans in executing the steps to improve compliance, consistency, and quality. That is, mass produce software applications like widgets in a manufacturing or factory setting. Generation 3 asynchronous technologies were a simplification and streamlining of earlier system lifecycles in that they focused on visual models like building blueprints which could be drawn up by architects. These blueprints could be checked for accuracy and automatically translated into pseudocode or the software source code itself (replete with automated test cases). Once again, their purpose was to codify the systems development process along with all of the systems design knowledge in visual representations.

Generation 4 asynchronous technologies were another simplification of systems development lifecycles. Many of the early agile frameworks were spinoffs of object-oriented methods. This was especially true for Feature Driven Development (FDD) which gained an early following as a bridge method between Generation 3 and 4 asynchronous technologies. However, early agile frameworks relied less on comprehensive systems modeling and visual design and reintroduced humans into the equation. Early agile frameworks suggested that humans should be given small batches of requirements, trusted and empowered to formulate simple system designs, and code, test, and deploy them quickly. They were based on the notion that the best and most creative programmer was the human using his or her own brain as a code generator, and that "Customers don't know what they wanted until they saw it!" That is, gather a few needs from the customers themselves; design, code, and test it quickly; show it to customers to gather their feedback; and rinse-and-repeat as often as necessary until customers are satisfied. Early agile frameworks relied on more tacit communications and teamwork than earlier asynchronous technologies, and small agile teams could converge on a solution in a few short weeks or months (versus decades and billions of dollars). Live, face-to-face synchronous communications, teamwork, cooperation, and collaboration were the key to early agile frameworks (however, antithetical to Western culture, psychology, and ideology).

Note. *One may ask, why are early agile frameworks considered an asynchronous technology given their reliance on synchronous communications. Well, for a number of reasons. Customer needs were often captured in the form of Class Responsibility Collaborator (CRC) Cards or User Stories, Product and Sprint Backlogs, and Release and Iteration Plans. Designs were often written on whiteboards, the code was all the documentation needed, and the unit tests were another manifestation of the software source code. These were all asynchronous forms of communication. Furthermore, early agile teams were very small, meaning that there was less dependence on high-speed social reciprocity or synchronous communication speed with hundreds or thousands of programmers and power-hungry governance boards and committees who walked and talked over people. Finally, early agile frameworks often used Pair Programming where a lead programmer was often teamed up with a novice programmer to create all code, cross skill, and communicate at the speed of neurodivergents (if at all). Oftentimes, the novice programmer simply watched the lead programmer code without interrupting. Finally, the novice programmer could go on to refactor, simplify, and streamline the code quickly and messily created by the lead programmer at a later time in an asynchronous fashion. All software code was stored in version control systems, and anyone could enhance, correct, or refactor the code at will on an as needed basis in an asynchronous fashion.*

Generation 5 asynchronous technologies really started to take shape with the use of instant messaging, email communications, blogs, bulletin boards, and even open-source software (OSS) developer communities. Developers across the world exchanged knowledge using blogs and bulletin boards 24 hours a day 365 days a year codifying worldwide software development knowledge and expertise. Blogs could also be used to capture short technical articles on key concepts, technologies, and recipes for solving problems (as well as promulgating emerging software development ideologies, frameworks, values, principles, practices, metrics, and tools). Instant messaging was used among collocated and geographically distributed programmers for asking questions, getting design advice, sharing code fragments, and debugging their code. Email could also be used as a form of asynchronous technology for reaching global developers who may or may not belong to a specific chat forum. But probably the most profound Generation 5 asynchronous technology was the emergence of the OSS community. They checked in their source code into global repositories for the world to see. Any programmer could download, improve, and upload the code for everyone to see. Millions of OSS developers worked alone in the confines of their houses, apartments, or bedrooms; communicated asynchronously; and the improved software source code was the ultimate form of human expression (no words necessary). Generation 5 asynchronous technologies were truly a mecca of neurodivergent programmers who were no longer constrained by the boundaries of high-speed social reciprocity and live face-to-face synchronous communications.

Generation 6 asynchronous technologies merged Generation 4 agile frameworks and Generation 5 asynchronous mediums together. As indicated earlier, these were represented by popular Agile ALM tools like Jira, Gitlab, TFS, Jazz, Confluence, etc. Agile ALM tools often codified the early agile frameworks themselves directly into workflow systems or were generic enough to be used for supporting one or more rudimentary agile frameworks. They were used for capturing product backlogs in the form of epics, features, user stories, tasks, and a variety of other issue types like defects. They were used to capture roadmaps, release plans, and sprint plans. They were used as primitive Scrum board, Scrumban, or Kanban systems to track user story status. They captured agile project metrics, models, measurements, and dashboards for performance reporting. They could capture the software source code itself in version control repositories; automated unit and acceptance tests; and serve as continuous integration (CI), continuous delivery (CD), and DevOps pipelines. And they could often capture management and developer communications, status messages, and any significant blockers or impediments. Tools like Confluence allowed direct integration with Agile ALM tools, visual project management plans and performance reports could be easily codified, shared, and reviewed; system designs and documentation could also be easily created, shared, and version controlled; and process assets and knowledge could also be created and shared across enterprises, portfolios, programs, products, and teams.

Note. *One important note was that 90% of global IT budgets were tied up in obsolete legacy systems. So, 9 out of 10 IT projects often involved repairing, maintaining, or improving legacy systems; legacy systems existed as enormously complex code monoliths; and it often took hundreds of programmers months and years to untangle them, make simple code changes, and reconstitute them into workable products. So, the planning and execution data in Agile ALMs was often enormous, complex, and voluminous to capture the intricate detail of modifying monolithic legacy systems. Keep in mind that early agile frameworks were designed to quickly create innovatively new products and services. They weren't necessarily designed for the formation of projects with hundreds or thousands of programmers, thousands of user stories, multi-year or multi-decade roadmaps and release plans, or agile projects with hundreds of iterations spanning years and decades. However, the sin of Agile ALMs was that their databases were bottomless pits of user stories without WIP limits into which traditional project managers were encouraged to dump their 15,000-line integrated master schedules (IMSs) or foot-high stacks of business requirement documents created by eager and highly motivated Big Six Consultants. But more on this conundrum in the next paragraph.*

Finally, Generation 7 asynchronous technologies significantly improve upon Generation 6 technologies which may end the neurodivergent wars once and for all (along with their irreconcilable legacy system conundrums). As we mentioned before, these technologies include, but are not limited to, ALM Tools like Asana, Ayanza, Clickup, Wrike, Height, Hive, Butler, Copilot, Jira ML, etc. These technologies have the ability to data mine IMS and business requirements to form product and sprint backlogs, release and sprint plans, estimate story points and task durations, and automatically forecast completion dates (in real time). It's sort of like your smartphone or electric vehicle forecasting traffic and weather conditions, begin and end points, congestion and obstacles, rerouting you for optimal travel times, and updating your final arrival time as you go along based on your progress and emerging highway conditions. Of course, it's necessary to classify the type of product or project timeline you're pursuing in order to calibrate the tool's heuristics for optimal performance. That is, are you performing a one-week design sprint; early business experiment; series of exploratory MVPs; accumulating monitoring, and forecasting innovation metrics to determine the optimal market strike point; operating and maintaining a production cloud; scaling a proven MVP into a production system; or, in the worst case, modernizing a monolithic legacy system. Each of these product or project types is vastly different and there is no one size fits all. Generation 7 asynchronous technologies also include Cloud-based AI and ML driven CI, CD, and DevOps pipelines for quick, inexpensive, and scalable application deployments. More importantly, they include communication mechanisms like reporting, dashboards, chatbots, AI assistants, data mining, and suggestions for improving communications, quality, and performance. For instance, it might help a programmer complete a code module quickly by identifying and inserting an optimal reusable component for quick application composition in seconds, minutes, and hours (as opposed to days, weeks, months, or years). It may also incorporate visual design information, developer narratives, conversations, and even video into the code itself (so that maintainers can see a history of how and why that code design was chosen, how to improve it, and its performance limitations).

Note. *But the best is yet to come. We still have the major problem that over 90% of global IT assets exist as monolithic legacy systems that are difficult to repair, maintain, update, or even migrate to clouds where their data can be preserved, reused, and data mined. For instance, let's say a merger and acquisition results on the inheritance of 200 monolithic enterprise applications with several decades worth of customer transactions. One option may be to migrate these legacy systems or databases to a shared cloud where the customer information may be stored, maintained, data mined, extracted, or loaded into one or more customer relationship management (CRM) systems. Better yet, AI and ML agents can quickly scan the software source code, database schemas, or live IT systems themselves; reverse engineer an as-is or as-built operational system architecture; identify breakpoints, interfaces, and fissures in which to modularize the legacy systems; apply an AI-driven strangler algorithm to quickly form service oriented architectures; or even quickly form small modularized microservices which can be replaced, repaired, or enhanced quickly. In one instance, it took a team of 300 people six months to change the location of a user interface element on a monolithic legacy system. With AI-driven reverse engineering, the code could be automatically decomposed and modularized into microservices, the user interface element could be repositioned, and the microservice based architecture could be retested, and redeployed in hours and days with a small team of developers.*

Summary. So, we've covered a lot of ground here. We've illustrated how the creation of the electronic computer emerged in the 1940s which implicitly leveled the playing field between neurotypes and neurodivergents. And we've illustrated how each successive wave or generation of asynchronous information technologies explicitly leveled the playing field between neurotypes and neurodivergents. We showed how the earliest asynchronous technologies sought to codify tacit (intangible) knowledge in the form of explicit (tangible) system lifecycles, processes, governance boards, tools, documents, tests, and workflow systems (i.e., software factories, CASE, Agile ALMs, etc.). We also showed how earlier asynchronous technologies sought to eliminate or replace tacit knowledge or synchronous communications altogether, while early agile frameworks sought to replace explicit artifacts with fast face-to-face synchronous communications. However, we also illustrated that early agile teams were small, cohesive, and just about the right speed for many neurodivergents. That is, neurodivergents often perform best when interacting with inanimate objects like ALM tools, release and iteration plans, Confluence pages, User Stories, design information, software source code, unit and acceptance tests, and DevOps pipelines. And, of course, neurodivergents perform best in small group and team settings as opposed to monolithic projects and products comprised of hundreds or thousands of people, conversations, and communication paths. Finally, we labored to make the case that the IT industry explicitly and implicitly labored to level the playing field between neurotypes and neurodivergents for 70 years who have widely disparate cognitive abilities and operating speeds which often manifest as cognitive and verbal delays. However, with slower systems development lifecycles and asynchronous technologies, the IT industry has done its best to minimize the cognitive distance necessary for IT success.

Agile IT Frameworks for Neurotypes vs. Neurodivergents

So, based on the earlier timeline, history, classification, and taxonomy of asynchronous IT technologies, let's attempt to break this down by which category and technology applies best to neurotypes vs. neurodivergents. Remember, that in general, neurotypes have much faster and wider cognitive abilities, while neurodivergents may have much slower and slightly deeper cognitive abilities. Neurotypes tend to have densely packed neural networks with high-speed interconnections, while neurodivergents tend to have sparse neural networks with much weaker and slower interconnections. The former leads to quick holistic decision making, while the latter leads to slower incomplete decision making. This is often referred to as Theory of Mind (ToM) or Emotional Intelligence (EI), where Neurotypes may be able to predict an outcome based on many past, present, and future variables in real-time, whereas neurodivergents are poor at predicting, anticipating, or planning for widely disparate conditions and outcomes in advance. Furthermore, they have cognitive and verbal delays, may not be able to compose quick, terse responses in real-time interspersed in large group communication settings, may need time to let new information matriculate through their neural network, and then compose an asynchronous response. That is, they may need some time to compose their thoughts in writing over a few days and broadcast it using email, written reports or technical notes, monologues, or long chains of text or chat messages. Therefore, neurotypes may be best suited for large meetings of extroverted executives and directors communicating in intense rapid-fire style, while neurodivergents tend to avoid large professional or social groups or meetings with extroverts, and prefer to answer email, text, or other asynchronous notifications. Asynchronous communication slows the thought stream down a bit, provides time to compose a thoughtful and thorough response, and launch a message without interruption by an extroverted or ADHD executive.

Phase	Technology	Neurotype	Neurodivergent	Rationale
4 - Agile -	**Epic MVP**	No	Yes	Lean, lightweight
	Roadmap	No	Yes	Lean, lightweight
	Story Map	No	Yes	Lean, lightweight
	Release Plan	No	Yes	Lean, lightweight
	Sprint Plan	No	Yes	Lean, lightweight
	Scrum Board	No	Yes	Visual, lean, light
	User Story	No	Yes	Lean, lightweight
	Design	No	Yes	Lean, lightweight
	Daily Scrum	No	Yes	Telegraph

	Blockers	No	Yes	Asynchronous
	F2F Coms	Yes	No	Cognitive Delays
	Test	Yes	No	Manual
	Sprint Review	No	Yes	Monologue
	Sprint Retro	Yes	No	Fast, Synchronous
	Refinement	No	Yes	Lean, lightweight
	SoS, PO Synch	No	Yes	Telegraph
	System Demos	No	Yes	Monologue
	Release Reviews	No	Yes	Telegraph
	Deployment	Yes	No	Manual
5 - Web Tech -	**Bulletin Board**	No	Yes	Asynchronous
	Email	No	Yes	Asynchronous
	Blog	No	Yes	Asynchronous
	Chat Room	No	Yes	Asynchronous
	Messaging Apps	No	Yes	Asynchronous
6 - Agile ALM -	**Legacy System**	Yes	No	Monolithic, Complex
	Roadmap	No	Yes	Lean, Asynchronous
	Story Map	No	Yes	Lean, Asynchronous
	Release Plan	No	Yes	Lean, Asynchronous
	Epic	No	Yes	Lean, Asynchronous
	Feature	No	Yes	Lean, Asynchronous
	Sprint Plan	No	Yes	Lean, Asynchronous
	Scrum Board	No	Yes	Lean, Asynchronous
	User Story	No	Yes	Lean, Asynchronous
	Design	No	Yes	Lean, Asynchronous
	Notifications	No	Yes	Asynchronous
	F2F Comms	Yes	No	Cognitive Delays
	Email	No	Yes	Asynchronous
	Text, Chat	No	Yes	Asynchronous
	Confluence	No	Yes	Lean, Asynchronous
	Daily Scrum	No	Yes	Telegraph
	Blockers	No	Yes	Asynchronous
	CI, CD, DevOps	No	Yes	One-Piece Workflow
	Build Reports	No	Yes	Asynchronous
	Test	No	Yes	Automated
	Test Reports	No	Yes	Asynchronous
	Sprint Review	No	Yes	Monologue
	Sprint Retro	No	Yes	Yes, Asynchronous
	Deployment	No	Yes	Automated
7 - AI ALM -	**Variable Scope**	Yes	Yes	Lean, Automated
	Roadmap	Yes	Yes	Lean, Automated
	Microservices	Yes	Yes	Lean, Automated
	Story Map	Yes	Yes	Lean, Automated
	Release Plan	Yes	Yes	Lean, Automated

	Epic	Yes	Yes	Lean, Automated
	Feature	Yes	Yes	Lean, Automated
	Sprint Plan	Yes	Yes	Lean, Automated
	Scrum Board	Yes	Yes	Visual, Automated
	User Story	Yes	Yes	Lean, Automated
	Design	Yes	Yes	Lean, Automated
	Notifications	Yes	Yes	Automated
	F2F Comms	Yes	Yes	Video, telegraph
	Email	Yes	Yes	Automated
	Text, Chat	Yes	Yes	Automated
	Confluence	Yes	Yes	Lean, Automated
	Daily Scrum	Yes	Yes	Automated
	Blockers	Yes	Yes	Automated
	CI, CD, DevOps	Yes	Yes	Automated
	Build Reports	Yes	Yes	Automated
	Test	Yes	Yes	Automated
	Test Reports	Yes	Yes	Automated
	Sprint Review	Yes	Yes	Automated
	Sprint Retro	Yes	Yes	Automated
	Deployment	Yes	Yes	Automated
	Production	Yes	Yes	Cloud, Automated

Again, agile frameworks are somewhat ideal for both neurotypes and neurodivergents. From a neurotypical perspective, agile frameworks enable ambitious and extroverted neurotypes to successfully conceive, deploy, and cash in on an innovatively new product or service in a short enough time to hit their quarterly OKRs and get promoted. Agile frameworks may not be well suited for repairing, maintaining, or improving a large monolithic legacy system (nor migrating dozens of them to a cloud in a short period of time in an effort to hoard their databases and data). Agile frameworks are well suited for neurotypical technology executives, product managers, and product owners to help manage mixed teams of neurotypical and neurodivergent programmers. Agile frameworks also have many benefits for neurodivergent programmers. They tend to favor small MVPs, simple designs, limited WIP, timeboxed events, minimal processes and documentation, short telegraph style communications, and more show and tell with working systems and software (vs. vaporware, dog-and-pony shows, and other marketing events were neurotypicals shine best). In summary, we've seen a shift from simply codifying large amounts of information in document libraries which take decades and billions of dollars (along with 30-ton computers), to small, simple, fast, asynchronous, automated, and AI-driven agile frameworks and ALMs. The biggest shift has been from building or maintaining impossibly large legacy monoliths with hundreds or thousands of people which take years or decades, to rapidly composing and deploying microservices in minutes and hours to global cloud fabrics using automated DevOps pipelines. The biggest innovation emerging, of course, is the ability of AI services to automatically scan, reverse engineer, visualize, and decompose enormously complex legacy system monoliths into smaller modularized microservices which can be containerized and deployed to billions of global end points in fractions of a second. And, of course, all of these capabilities are available to small groups of teams, pair programmers, and individual developers where the risks of cognitive and verbal delays are minimized, managed, or mitigated.

Current Agile AI Tool Examples (which optimize asynchronous communications for small teams)

Basic Management	Project Management	**ScrumDesk** - Automates OKRs, story mapping, product backlogs, sprint planning, collaboration, and reporting.
	Product Management	**Archy** - Generates epics, stories, and acceptance criteria from industry, historical, and team performance.
	Dashboard Management	**QueryCraft** - Automatically generates JQL queries from natural language descriptions for ALM reporting.
	Performance Management	**Ogoodo** - Timeline predictions, automated metric tracking, workflow optimization, and delivery management.
	Design Management	**Figflow** - Converts Figma user interface UX wireframes and designs into user stories for product teams.
	Requirements Management	**Klofa** - Automatically generates epics and user stories during online product meetings.
	Story Management	**Stoorai** - Helps Product Owners automatically generate user stories from titles.
	Task Management	**GeniePM** - Automatically creates user stories, tasks, and descriptions.
	Collaboration Management	**Swift Board** - Anonymous brainstorming, sprint planning, and real-time team collaboration.
	Retrospective Management	**MyPop** - Provides real-time insights and improves project outcomes from multiple collaboration tools.
Technical Management	Performance Management	**Stepsize AI** - Automatically generates sprint reports, performance analytics, and tracks key trends.
	DevOps Management	**Plandek** - Automatically data mines, tracks, and reports DevOps performance data from multiple ALM tools.
	Meeting Management	**Otter AI** - Automatically transcribes online meetings, conversations, meeting minutes, and action items.
	Retrospectives	**Retrium** - Automatically manages structured and unstructured retrospectives, tracks data, and identifies patterns.
	Brainstorming	**Miro AI** - Automates brainstorming, strategy formation, collaboration, rapid prototyping, and quick retrospectives.
	Workflow Management	**Jira Align AI** - Automatically analyzes the performance and bottlenecks of larger scaled agile projects.
	Sentiment Management	**Team Mood** - Automatically generates sentiment, engagement, and collaboration from asynchronous platforms.
	Estimation Management	**Taskade** - Automatically generates reliable task estimates based on historical, team, and individual data.
	Acceptance Criteria	**Easy Peasy** - Automates the creation of acceptance criteria for epics, features, and user stories for teams.
	Definition of Done	**Notion AI** - Automates creation, testing, consistency, and management of documenting the Definition of Done.
	Scrum Management	**ChatGPT** - Acts as a virtual Scrum Master assistant by offering real-time suggestions, insights, and hints.
	Writing Assistant	**QuillBot** - Helps Product Owners, Scrummasters, and Teams form clear, polished, and professional communications.
Software Development	Data Science	**DataRobot** - Automated machine learning platform that builds and deploys predictive models swiftly.
	Project Management	**Unito** - Collects, integrates, and centralizes project management data from multiple tools for agile tracking.
	Story Management	**Leiga** - Automated user story and task tracking, management, and reporting to help developers stay productive.
	Coding Assistant	**Kite** - Low level coding assistant that generates pseudocode patterns and hints as programmers type statements.
	Coding Proofreader	**DeepCode** - Automated software source code static analysis that identifies and corrects early coding errors.
	Bug Tracking	**BugBot** - Automated test generation, execution, recording, data mining, and bug tracking for complex systems.
	Code Refactoring	**Refraction.dev** - Automated code refactoring, test generation, and source code documentation generation.
	AI Monitoring	**Aporia** - Automated performance and security monitoring of built-in AI components, utilities, and applications.
	General Monitoring	**Dynatrace** - Tracks customer behavior, application performance, infrastructure monitoring, and user experience.
	Task Automation	**UiPath** - Automates administrative tasks, forms, activities, and customer workflows to focus on value adding work.

Sources. https://www.eliteai.tools • https://www.scrum.org • https://www.leiga.com

Summary

Okay, so what have we learned in this short treatise on agile information technologies for neurotypes and neurodivergents? Well, we've learned that neurotypes have extremely fast, wide, and deep cognitive speeds and abilities, while neurodivergents have substantially slower, narrower, and shallower cognitive abilities. This often manifests itself in the form of cognitive and verbal delays (and a broad array of other comorbidities). These comorbidities may include substantially increased anxiety in the face of complexity and uncertainty, stress, selective mutism (silence), elopement (departure), catatonia, isolation, introversion, obsessive compulsiveness, stimming, addiction, manic depression, bipolarism, and even fits of anger, rage, and frustration. Stimming is also known as stereotypies which are defined as nonsensical repetitive movements or vocalizations (i.e., tremors, hand flapping, repetition, spinning, banging objects, head banging, self-injury, etc.). However, agile frameworks and asynchronous technologies have come a long way to minimizing complexity and uncertainty, which reduces the anxiety and occurrence of suboptimizing comorbidities. Agile frameworks do this by limiting the scope of agile products to simple Epic-MVPs, business experiments, and other small architectural runways (or foundational elements). Furthermore, agile frameworks suggest the development of small virtual cloud-based vertical slices vs. monolithic physical brick-n-mortar layers with long lead times. That is, Epic MVPs that can be rapidly composed, deployed, and evaluated to a global cloud fabric to millions and even billions of end users in seconds, minutes, hours, and days. Agile frameworks are simple, short, timeboxed, and involve one-way telegraphic communications (i.e., what did you do yesterday, what will you do today, and do you have any blockers). Neurodivergents are famous for preplanning well-scripted communications, so Daily Scrums are an ideal communication forum for them. Furthermore, agile frameworks involve a large toolbox or Swiss Army Knife of asynchronous communication technologies such as Agile ALMs, Confluence pages, Roadmaps, Epics, Features, Story Maps, Release Plans, Sprint Plans, User Stories, Tasks, Scrum Boards, Sprint Reviews, Sprint Retrospectives, Automated Unit and Acceptance Tests, DevOps Pipelines, and Automated (Scripted) Cloud deployments. Better yet for neurodivergents, is that agile frameworks demand small self-organizing agile teams, where neurodivergents communicate best with short telegraphic communications, monologues, email messages, text messaging, whiteboarding, automated ALM notifications, test and build reports, etc. AI-driven ALMs further automate the construction of Agile roadmaps, epics, features, story maps, stories, release plans, sprint plans, ceremonies, asynchronous reporting, architecture and design, software composition, automated testing, DevOps, deployment, security, monitoring, performance measurement, and even end-user experience, satisfaction, and growth. Finally, AI-driven ALMs solve the global legacy system monolith crisis, reverse engineer their architectures, recommend code modularization, automate strangulation, form service-oriented architectures, fine tune microservices, containerize them, and deploy them to staging and production fabrics for further performance analysis, optimization, and end-user evaluation. What we've seen over the last 70 years is not a sudden adaptation of agile frameworks, ALMs, and AI for a neurodivergent workforce, but a subtle and gradual evolution of information technology for neurodivergents precipitated by the vast cognitive differences between neurotypes and neurodivergents from the earliest days of computer programming in the form of AI-driven Agile ALMs with asynchronous communication mechanisms.

Endnote—William Whyte's 50 Cent Spindle

William Whyte is credited with creating an ingenious method for asynchronous communications between neurotypes and neurodivergents in the 1940s. It just so happens that the city of Chicago found itself in a rather precarious position following World War II. Men of fighting age were drafted into World War II while U.S. women were hired into businesses to backfill the dearth of male labor. However, upon the war's end, men returned to find businesses chock full of female workers. The pecking order had certainly been disrupted. Violent confrontations immediately ensued and before you know it, Chicago area restaurants were at a standstill.

The city of Chicago enlisted the help of William Foote Whyte who had a newly minted PhD in Sociology from the University of Chicago. Although trained in state-of-the-art methods in psychology, sociology, and anthropology, the torrent of violent outrage was simply too much for him to manage alone. Desperate to stop the fighting he was open to any suggestion. His research assistant, Edith Lentz, noted that some restaurants used a metal spindle to mediate communication between waiters and cooks. Whyte accepted her proposal, piloted it in some restaurants, scaled it after trial and error, and staved off the crisis. This was a lesson in "Lean Startup," "Lean Impact," "Lean Inception," or "Experimentation" (i.e., start small and scale up if you have favorable results).

The notion is that neurotypical waiters and waitresses who are cognitively faster record customer orders on a slip of paper and place them on the metal spindle. Neurodivergent cooks who are cognitively slower remove the slips on a first in first out (FIFO) basis one at a time (i.e., one-piece-workflow-system). There is no human-to-human communication or contact between neurotypes and neurodivergents (i.e., only a small window or counter separating the dining area from the kitchen in which to exchange plates for orders). It's an asynchronous communication technology to mediate the cognitive distance between neurotypes and neurodivergents.

Although Whyte did not invent the spindle, it was only a stopgap measure to give him time to enlist the values, principles, and practices of his sociology training. However, in the end, the stopgap measure—Metal spindle—Or asynchronous communication device turned out to be the solution to the new cognitive war. With today's smartphones and Internet, customers enter their own orders on electronic tickets, place them in electronic queues, and pick up their orders or have them delivered with little to no human interactions. ***Bottomline****? Asynchronous technologies are a good means of mediating communications between neurodivergent humans.*

Note. *It's interesting to note that early "Software Factories" enlisted the help of Whyte to help mediate cognitive communication issues between programmers designing early application lifecycle management (ALM) tools.*

Whyte, W. F. (1949). The social structure of the restaurant. American Journal of Sociology, 54(4), 302-310.

Limitations

Neither the American Psychological Association (APA) nor the global neurodivergent community agree on a definition of neurodivergence. Furthermore, the neurodivergence community disputes the difference between subclinical and clinical neurodivergence. As such, the global neurodivergence community disputes phrases like, "On the Spectrum," or that there is a dichotomy between neurotypes and neurodivergents. Furthermore, the neurodivergent community disputes the notion that neurodivergence is a condition, disease, or suboptimal cognitive state which must be treated, corrected, or managed. While there is some evidence that the human brain is remarkably neuroplastic (flexible) and may be reprogrammed or fine-tuned—and that structured and disciplined techniques such as Applied Behavior Analysis (ABA)—may be used for shaping or reshaping neurodivergent behavior, neurodivergents do not believe it's necessary to do so. The question then becomes do agile frameworks, asynchronous communication technologies, and AI-driven ALMs constitute an ABA technique or a simple adaptation to neurodivergent cognitive speed and capability. We'd argue the latter. That is agile frameworks, asynchronous communications, and AI-ALMs help neurodivergents contribute to successful IT projects without reprogramming them. Now, if a neurodivergent wishes to behave more neurotypical or function optimally in a neurotypical setting, then perhaps some ABA behavioral shaping, reprogramming, rewiring, or conditioning may be in order. Hence, we argue that as is, agile frameworks, asynchronous technologies, and AI-ALMs constitute a natural instinctual expression of the neurodivergent cognitive state, speed, and ability. Keep in mind that we are not trained nor certified medical, psychiatric, psychological, or other mental health care professionals. The views expressed in this article are solely those of the authors and do not reflect the official policy or position of the medical, healthcare, or neurodivergent community. The content of this treatise is for informational purposes only and is not intended to diagnose, treat, cure, or prevent any condition or disease. Readers understand that this treatise is not intended as a substitute for consultation with a licensed practitioner. Please consult with your own physician or healthcare specialist regarding the suggestions and recommendations made in this treatise. The use, consumption, and redistribution of this treatise implies your acceptance of this disclaimer.

Further Reading

- Amen, D. G. (2013). Healing ADD: The breakthrough program that allows you to see and heal the 7 types of ADD. New York, NY: Berkely Books.
- Beck, K. (2000). Extreme programming explained: Embrace change. Reading, MA: Addison-Wesley.
- Beck, K., & Fowler, M. (2001). Planning extreme programming. Upper Saddle River, NJ: Addison-Wesley.
- Brechner, E. (2015). Agile project management with kanban. Redmond, WA: Microsoft Press.
- Carter, V. (2024). Scrum-tious: ChatGPT for product owners and scrum masters. Philadelphia, PA: Carter-Evans Publishing.
- Chang, A. M. (2019). Lean impact: How to innovate for radically greater social good. Hoboken, NJ: Wiley & Sons.
- Johnson, C., & Crowder, J. (1994). Autism: From tragedy to triumph. Wellesely, MA: Branden Publishing.
- Knapp, J. (2016). Sprint: Solve big problems and test new ideas in just five days. New York, NY: Simon & Schuster.
- Leffingwell, D. (2023). Scaled agile framework (SAFe) 6.0. Boulder, CO: Scaled Agile, Inc.
- Lovaas, O. I. (1981). Teaching developmentally disabled children: The me book. Austin, TX: Pro Ed.
- Lovaas, O. I. (1997). The autistic child: Language development through behavior modification. New York, NY: Irvington Publishers.
- Lovaas, O. I. (2003). Teaching individuals with developmental delays: Basic intervention techniques. Austin, TX: Pro Ed.
- Maurice, C. (1993). Let me hear your voice: A family's triumph over autism. New York, NY: Random House.
- Maurice, C. (1996). Behavioral intervention for young children with autism: A manual for parents and professionals. Austin, TX: Pro Ed.
- Pichler, R. (2010). Agile product management with scrum: Creating products that customers love. Upper Saddle River, NJ. Addison-Wesley.
- Pizano, G. (2024). Clear impact: Strategic AI for game-changing product management. Richmond, VA: Transformative Tracks & Learning.
- Pizano, G. (2024). Empowered by AI: A scrum master's guide to next-level agility. Richmond, VA: Transformative Tracks & Learning.
- Rajwanshi, G. (2025). AI powered scrum master: Redefining agile leadership through AI. Slough, UK: Advance Agility.
- Reddy, A. (2016). Scrumban revolution: Getting the most out of agile, scrum, and lean kanban. New York, NY: Addison-Wesley.
- Ries, E. (2017). The startup way: How modern companies use entrepreneurial management to transform culture and drive long-term growth. New York, NY: Crown Publishing.
- Rupp, C. G. (2020). Scaling scrum across modern enterprises: Implement scrum and lean-agile techniques across complex products, portfolios, and programs in large organizations. Birmingham, UK: Packt Pub.
- Sammicheli, P. (2022) Scrum in AI: Artificial intelligence agile development with scrum and mlops. Siena, SI: GPTplane.
- Schrage, M. (2014). The innovator's hypothesis: How cheap experiments are worth more than good ideas. Boston, MA: MIT Press.
- Schwaber, K. (2004). Agile project management with scrum. Redmond, WA: Microsoft Press.
- Schwaber, K., & Beedle, M. (2001). Agile software development with scrum. Upper Saddle River, NJ: Prentice-Hall.
- Schwaber, K., & Sutherland, J. (2020). The scrum guide: The definitive guide to scrum (the rules of the game). Mountain View, CA: Creative Commons.
- Schwartz, J. M., & Begley, S. (2002). The mind and the brain: Neuroplasticity and the power of mental force. New York, NY: Harper Collins Publishers.
- Thomke, S. H. (2020). Experimentation works: The surprising power of business experiments. Boston, MA: Harvard Business Review Press.
- Wilmshurst, D. (2023). SAFe coaches handbook: Proven tips and techniques for launching and running SAFe teams, ARTs, and portfolios in an agile enterprise. Birmingham, UK: Packt Publishing.

Actual Case Studies

Product Owner. *This was a decade long legacy product modernization effort. Agile teams were allowed to self-organize and select product owners and scrummasters. However, agile leadership decided to insert their own product owners and scrummasters into each agile team. Agile leadership implicitly believed that consistent agile practices were the key to speeding up product delivery (completion). A career agile coach and scrummaster was appointed to be a product owner, if not the lead product owner. The agile coach had been a product owner and scrummaster in alternating cycles for over two decades. The agile coach was a neurodivergent who relied upon asynchronous ALMs and other support tools to participate, facilitate, and lead agile teams. Early in her career, the coach only worked on agile teams without the aid of Agile ALMs. However, as the popularity of Agile ALMs grew, she made it a point to quickly learn, master, and apply as many Agile ALMs and collaborative asynchronous platforms as possible. She simply didn't want to be left behind and many of her clients needed more Agile ALM training than agile framework training by this point in her career. She enjoyed the Agile ALMs and collaborative platforms this agile product team applied. Agile leadership teams implicitly believed in the power of asynchronous agile technologies for planning and managing largescale product teams. As product owner, she quickly identified or formulated new epics in the Agile ALM, developed story maps to refine epics, formed 90-day release and sprint plans, and conducted strictly timeboxed agile events using virtual conferencing tools. She relied on neurotypical product managers and architects to inform her agile plans. She also sent Agile ALM performance reports to product managers. She really enjoyed the asynchronous nature of Agile ALMs, instant messaging, email, etc. She often appeared more neurotypical than software developers.*

Scrummaster. *This was also a decade long legacy product modernization. A scrummaster joined a team of backend engineers building an on-premises cloud. The backend teams had a part-time remote scrummaster who did his best to instill agile practices from a distance. The backend team provided operations and maintenance, services, and help desks for a large portfolio of enterprise IT legacy products. Although they were building a new on-prem cloud, they did not see the value in agile frameworks for reducing uncertainty and risk. The new scrummaster found the rudimentary fragments of agile frameworks, practices, tools, and teams. The backend teams self-organized and Frankensteined their agile structure without any formal training or experience. The backend teams were exceptionally talented and created some rather innovative agile practices and adaptations. She wasn't allowed to add any new agile ceremonies but had to integrate product backlogs from 6 or 7 disparate ALMs. She accepted that as her challenge to synchronize the data from the disparate ALMs into coherent plans. The backend teams refused to downsize to only one ALM and insisted on using all of them simultaneously. This was an odd adaptation since the backend teams were a bit small. She instantly formed coherent release and sprint plans, directly linked as many data sources as possible, and maintained a virtual traceability matrix in her head. Agile leadership teams immediately recognized the impact she made as backend teams hadn't submitted any agile plans until she arrived. They simply didn't understand how to form them. She sent Agile ALM performance reports to product managers and enjoyed the asynchronous nature of Agile ALMs, instant messaging, email, etc. which helped backend team focus on value adding work. She was the only neurodivergent in a sea of neurotypes although there were a handful of developers who were a bit more neurodivergent than she was.*

Developers. *This too was a decade long legacy product modernization. A team of cloud engineers were building containerization services for an on-premises cloud. This was a premium IT firm as cloud computing was black magic to the enterprise. The cloud firm hired neurodivergents in droves and its leader was very neurodivergent. Was cloud computing beyond the cognitive abilities of neurotypes? A scrummaster was hired to help herd the cats comprising the cloud team. They did what they thought was best, in whatever manner they preferred, and at their own pace. They often sidestepped the enterprise's cybersecurity policies. The scrummaster had loads of cloud training and certifications. And she successfully rolled out agile frameworks on cloud teams for 15 years. She didn't comprehend the scope of what they were doing at first since their work existed as tacit knowledge. And the cloud team didn't write down what they did for fear of losing control of their intellectual capital. She was responsible for forming agile release and sprint plans for the cloud team, populating and operating Agile ALMs, and facilitating agile ceremonies as well. She pieced together as much as she could from informal tacit communications although the cloud team was reticent to divulge too much detail. Eventually, the cloud leader divulged the cloud architecture in an outline form which the scrummaster was easily able to convert into epics, features, stories, releases, sprints, and ALM entries. In other words, the cloud outline was a crude set of story maps. The scrummaster learned the value of story maps and took them to each new team she joined. The visual story maps were a boon to the cloud team and helped them deliver quickly and consistently. She sent Agile ALM performance reports to product managers and applied asynchronous communications to help manage the neurodivergent cloud team. Although a neurodivergent herself, the cloud team treated her as a neurotypical outsider.*