33 Principles and Practices for Improving Scaled Agile Framework (SAFe) Developer Experience (DX)

Tra·di·tion·al • De·vel·op·er • Ex·pe·ri·ence (*trə-dĭsh'ə-nəl • dĭ-vĕl'ə-pər • ĭk-spîr'ē-əns*). The ease-of-use of a software development language, tool, environment, lifecycle, office spaces, company culture, or company policies. In the early days of computer programming (i.e., 1950s) there was a space race to create human readable computer programming languages like FORTRAN, FLOWMATIC, COBOL, etc. The theory was that if the computer programming language closely resembled a natural language like English, then it would ease the task of programming complex computers. Traditional developer experience expanded into compilers, editors, debuggers, configuration management tools, project management tools, and early software development environments. There was a dearth of computer programmers from the 1950s to the 1970s, and the belief was that the easier it was to create complex computer programs, the easier it would be to meet the explosive demand for such a technical field. By the 1980s, computer aided software engineering (CASE) environments were all-the-rage, purporting to automate the entire software development lifecycle process, including project management, requirements analysis, architecture, design, programming, testing, configuration management, etc. These evolved into integrated development environments (IDEs) in the 1990s and 2000s, with competitors like Microsoft's Visual Studio and IBM's Eclipse for Java. A myriad of project management and software development lifecycles emerged from the U.S. DoD, public sector agencies, IEEE, ISO, and variety of private sector organizations. Although most high-level computer programming languages, software development tools and environments, and project management and software development lifecycles were created to help make software development easier, they often had an inverse effect, making software development slower, more bureaucratic, and painful (i.e., anything but easier for developers).

Ag·ile • De·vel·op·er • Ex·pe·ri·ence (*ăj'əl • dĭ-vĕl'ə-pər • ĭk-spîr'ē-əns*). Simplified, rapid development lifecycles and technologies. By the 1990s, developer experience took a giant leap forward after 50 years and billions of dollars in public and private sector investments in making software development easier and more productive. This started with widespread use of C and C++, advent of DOS and Microsoft Windows, and high-performance low-cost WinTel PCs. Visual Basic, PowerBuilder, and HTML were born in the 1990s when the WWW exploded onto the global scene. Rapid Application Development (RAD), Joint Application Development (JAD), Participatory Design (PD), and Rational Unified Process (RUP) became popular. The Unified Modeling Language (UML) was the preferred design notation, and then agile methods suddenly burst onto the global scene instantly stripping away 50 years of amalgamated rust all at once as exemplified by capability maturity models from the 1960s mainframe era. Suddenly, Scrum, Extreme Programming (XP), Feature Driven Development (FDD), Dynamic Systems Development Methodology (DSDM), Crystal Methods, and others became very popular. They were simplified software lifecycles that could be combined with the latest software development tools on low cost WinTel PCs. All one had to do was identify a top 10 list of features, start programming, test and deliver, and endlessly rinse-and-repeat every two to four weeks. Instantly blown away was the need for 5-10-or-15-year integrated master schedules (IMSs), libraries of documents, and armies of project bureaucrats. It was the perfect paradigm for developers at Netscape releasing new browsers to the Internet in 24-hour cycles (and short-term throwaway experiments were acceptable alternatives to 50-year gold plated products that absolutely no one ever used at all).

SAFe • De·vel·op·er • Ex·pe·ri·ence (*sāf • dĭ-vĕl'ə-pər • ĭk-spîr'ē-əns*). A much-needed simplification of lean-agile frameworks for larger projects and product ecosystems that have been highjacked and misapplied for creating IMS-driven feature factories and sweatshops. While simple agile frameworks such as Scrum and XP were very effective for small teams rapidly evolving small business experiments with short shelf lives, it was clear that even these teams grew in size as innovative products and services evolved into production systems. That is, software development projects grew into dozens of teams with business success and needed more project coordination than Scrum and XP were designed to support. Agile project management methods emerged in the mid-2000s to begin addressing the perceived gaps and risks of scaling agile methods up to teams of teams. The Scaled Agile Framework (SAFe) emerged around the same period and finally hit the market around 2011 as the preferred agile framework for larger projects and even groups of projects (i.e., portfolios). SAFe emerged as a multidimensional service product replete with an industrial strength title, intuitive graphical model, portal or online wiki of SAFe knowledge, training and certification, and supporting textbooks. It was the perfect storm as agile methods finally penetrated mainstream consciousness, the Internet stabilized after threatening to go belly up in 2000, and deep-pocketed Fortune 500 firms wanted to instantly apply lean-agile principles and practices to modernize their infrastructures with digital transformation initiatives. SAFe itself continued evolving to become the de facto lean-agile framework and embodying more and more lean thinking values and principles, while adapting as many agile best practices as possible, including continuous integration, continuous delivery, DevOps, Lean UX, Kanban, and lean startup ideas.

While SAFe evolved into a state-of-the-art lean-agile thinking framework for business agility, portfolio management, large systems, and moderately sized projects, enterprises continue to misapply SAFe rather severely. Fortune 500 firms and public sector agencies use SAFe to create lean, agile, and traditional hybrids to serve as fully utilized IMS-driven feature factories. That is, firms continue to produce 5-10-and-15-year IMSs, enterprise architectures, and business requirements and plan out every program increment (PI) down to the minute for several hundred people over the course of years. They devise a one-time big design up front (BDUF) without lean-agile product management nor lean UX and apply divide-and-conquer principles to create complex portfolio, large solution, and program backlogs. Furthermore, BDUF teams create backlogs down to the user story level, formulate PI objectives in advance, and throw these over the wall the day of PI planning. Then they baseline user stories and story points in rigid agile application lifecycle management (ALM) tools, and instantly punish any deviation from EVM norms (while reserving the right to add new scope to team backlogs at any time). Afterall, SAFe is an adaptable lean-agile framework, right? Worse yet, 80% to 90% of teams on solution and agile release trains are functional administrative teams, and SAFe implementations devolve into non-stop all-day meetings to resolve dependencies and count story points as often as possible to satisfy accountants, schedulers, and other bean counters. This high-tech sweat shop IMS-driven feature factory approach must end as SAFe continues to evolve into a simplified business experimentation framework. Let's examine a few principles and practices to convert your high-tech IMS-driven feature factory sweatshop into the profitable, moderately paced, and enjoyable innovation team it is intended to be.

SAFe DX Principles & Practices

1. Lean-Agile • Think·ing (*lēn-ăj'əl • thĭng'kĭng*) Thin, slim, trim, light, fast; To apply small batches of WIP-limited business experiments to gradually tease out tacit, inexpressible requirements for innovatively new products and services

   ✓ Customer and market focused.
   ✓ Fast, waste-free delivery.
   ✓ Value stream mapping.

   The single most important principle of SAFe DX is clearly the notion of lean-agile thinking. That is, teasing out tacit, hidden, and inexpressible customer, market, and end-user needs with small batches of WIP limited business experiments. Traditional thinking is based on the notion of filling in every conceivable minute, hour, and day with some unneeded activity using integrated master schedules (IMSs), business requirements, and enterprise architectures in a vain attempt to squeeze every ounce of efficiency from the workforce. Full utilization is the antithesis of innovation, therefore lean-agile thinking introduces the concept of not-so-full-utilization in order to leave room for innovative thought, activities, and solutions. The scope of the solution or ecosystem of solutions should be as narrow (lean) as possible, a few small (lean) business experiments formed to tease out validate requirements, and a (lean) minimum viable product (MVP) instead of an over-scoped, gold-fleeced, or gold-plated solution requiring decades and billions of dollars that no one needs. When one reduces the scale, scope, and magnitude of the effort, process, and solution-set, then not only is lightning-fast innovation unleashed and user experience for the market, customers, or end-users enhanced, but the overall user experience for the innovators themselves is improved. One of the pillars of the lean-agile thinking is “respect for people” and lean-agile thinking embodies that pillar by not only respecting the voice of the customer but respecting the innovators themselves enough to give them an opportunity to succeed with lean thinking systems and solutions. We’re just now beginning to discover that a happier workforce leads to happier customer, and conversely, a dissatisfied workforce leads to a dissatisfied customer. However, happier innovators and customers do not come from fully burdened IMSs, business requirements, and enterprise architectures used to squeeze blood out of a turnip, but rather sparsely populated lean-agile systems, business experiments, and the solutions themselves. Speed is not achieved by forming overburdened feature factories, which many lean-agile thinkers are tempted to do. That is, they peak utilization to maximum velocity and then program this velocity into a long-term fully burdened IMS or product roadmap to cram in as many epics, features, user stories, and tasks as possible because this behavior is what traditional thinking rewards. Furthermore, no business experiments are planned, as epics, features, user stories, and tasks are a one-time, one-and-done deal. This means that the future is predicted (incorrectly), codified and delivered, and markets, customers, and end-users are forced to use unneeded, unwanted, and non-innovative solutions. Instead, lean-agile thinking involves hypothesis tests, rinse-and-repeat cycles are used to validate market needs, and then a final solution is codified after a few cycles. Of course, fully utilized sweatshop feature factories tuned to maximum velocity are not sustainable for innovators nor customers.

2. Lean-Agile • Lea·der·ship (*lēn-ăj'əl • lē'dər-shĭp'*) Guide, steer, point, inspire, influence; To exemplify, promulgate, and reward principles of lean thinking at all levels of an enterprise, portfolio, systems of systems, program, team, and individual level

   ✓ Exemplifies lean thinking.
   ✓ Rewards lean-agile behaviors.
   ✓ Trusts lean thinking completely.

   Of course, the second most important principle of SAFe DX is lean-agile leadership at all levels of an enterprise, portfolio, system of systems, small ecosystem of products and services, or even a small product or service itself. What this means is that not only should the top-level leadership exemplify, embody, promulgate, and reward lean-agile behaviors, but everyone in the leadership chain down to the lowest levels. Yes, this includes the frontline innovators themselves. There’s no sense in establishing a lean-agile leadership network only to have the innovators fighting lean-agile thinking tooth and nail. Therefore, enterprises must not only proactively invest in a lean-agile leadership network from top to bottom, but they must invest in instilling lean-thinking principles in the frontline innovators. Some principles that the leadership network must embrace should be getting trained, educated, experienced, and certified in lean-agile thinking principles, practices, and frameworks. Leaders should promulgate and expect lean thinking principles, practices, and frameworks to be applied. Leaders must reward lean-agile behaviors, and, most importantly, they must exemplify and embody the lean-agile thinking behaviors themselves. Leaders must not be hypocrites, contrarians, or be unwilling to eat their own dogfood. For instance, leaders should not devise integrated master schedules (IMSs) to rollout a lean-agile framework. That would be nonsense. Furthermore, true lean-agile leaders should not exhibit contrarian behaviors, like cooking up lean, agile, and traditional hybrids that include IMSs, earned value management (EVM), enterprise architectures, business requirements, and a host of debilitating traditional thinking practices like traditional capability maturity models and long-lead item technologies. Lean-agile leadership focuses on extremely small batches, small frequent business requirements, extra capacity for innovation, and rapid feedback cycles. None of these goals can be easily accomplished with long lead items. Technology simply changes too fast to build innovative products and services with long lead item technologies and solutions. Lean-agile leadership means precisely that, to exhibit lean-agile leadership. Furthermore, many traditional thinking principles and practices such as IMSs, EVM, business requirements, enterprise architectures, long lead item development, and capability maturity models originated in the public sector. However, today, most public sector agencies have gotten out of the business of legislating traditional thinking delivery models and are now embracing lean-agile thinking principles, practices, and frameworks. Therefore, if your enterprise is still promulgating traditional thinking principles, practices, and frameworks, or advocates hybridization of lean, agile, and traditional views, then it is simply starved for true lean-agile leadership. So, do yourself a favor, ditch traditional principles, practices, and frameworks, along with hybridization, and embrace lean-agile thinking for the sake of your long-term success.

3. Lean-Agile • Frame·works (*lēn-ăj'əl • frām'wûrkz*) Support structure, skeletal enclosure, scaffold platform, broad architecture; To apply proven highly-cohesive principles, practices, and reference models to create innovatively new products and services

- ✓ Proven lean-agile framework or method.
- ✓ Embodies lean-agile values and principles.
- ✓ Cohesive system of ceremonies and practices.

Another closely interrelated principle of SAFe DX is obviously the identification, selection, and proper application of lean-agile frameworks. There are many out-of-the-box lean-agile frameworks from which to choose, such as Scrum, Scrumban, Kanban, XP, FDD, Crystal Methods, DSDM, DevOps, DevSecOps, Design Sprints, Lean Startup, Startup Way, Lean UX, etc., for smaller scaled products and services. And, of course, there are a growing number of lean-agile frameworks for larger scale solutions, such as SAFe, DaD, S@S, Enterprise Scrum, LeSS, RAGe, SAGE, Spotify, etc. When creating a single small product or service any number of small-scale lean-agile frameworks will suffice (i.e., Scrum, Scrumban, XP, DevSecOps, Design Sprints, Lean UX, etc.). However, when creating small ecosystems of closely interrelated products and services, then a larger-scale framework may be advisable (i.e., SAFe, S@S, etc.). The keyword or key phrase here is "small ecosystem!" True to lean-agile thinking, lean-agile leaders should strive to keep batch sizes, business experiments, and the scope and size of product and service ecosystems as small and sparsely populated as possible. There's a goldilocks zone, sweet spot, or center of percussion that's a little bit tricky to find when applying lean-agile thinking principles, practices, and frameworks for successfully creating innovatively new products and services. Small scale lean-agile frameworks simply do not grow beyond a single team very effectively and even larger scale lean-agile frameworks seem to break down quickly if there are too many teams. Of course, there's always the traditional temptation to use small, medium, and large-scale lean-agile frameworks as feature factories to operate at maximum velocity to burn off a 5-10-or-15-year integrated master schedule (IMS), business requirements specification, enterprise architecture, etc. Worse yet is that enterprises will often try to "roll-their-own" lean-agile framework from a myriad of somewhat incompatible lean-agile practices. Unfortunately, some enterprises will simply select the wrong lean-agile framework that is simply a conceptual textbook approach without many field trials or practical capability. So, choose wisely, aim for a proven lean-agile framework, don't try to scale a small model for more than one team, don't select an impractical paper model, and don't go overboard and stretch a larger scale lean-agile framework beyond its known boundaries. It's not unheard of to see 500, 1,000, or even 2,000 person teams or greater applying a hodgepodge of inconsistent lean-agile principles, practices, and frameworks to create a monolithic product or service innovation on a shoestring budget. Chances are, larger initiatives are feature factories to realize over-scoped plans, requirements, and IMSs. Wanna succeed with lean-agile frameworks, aim for the smaller end of the scale (30, 50, or 70 people for larger ecosystems)? This, of course, is not to say that a single Scrum team of 5 to 10 people can't produce something extremely innovative!

4. De·cen·tral·ized • De·ci·sions (*dē-sĕn'trə-l-īzd' • dĭ-sĭzh'ənz*) Delegate, permit, endow, entitle, authorize; To empower front-line teams and individuals with the authority to select day-to-day options, choices, directions, preferences, and alternatives

- ✓ Bottoms-up decision-making rules.
- ✓ Day-to-day frontline decisions.
- ✓ Fastest possible innovation.

A critically important principle of SAFe DX is the practice of decentralized decisions and decision-making. That is, pushing day-to-day operating decision down to the lowest possible level. This, of course, isn't to say that there isn't an overarching set of guardrails in place in for the form of lean-agile values, principles, frameworks, practices, measurements, tools, and technologies. For instance, if your lean-agile framework is Scrum, then teams and individuals within those teams are expected to do Sprint planning, daily standups, demonstrations, retrospectives, etc. However, the Sprint plan, Sprint goals, work items such as user stories, and performance measures such as burndowns, velocity, and story points belong to the teams. The Scrum team's commitment it to the goal, not the user story, story point, nor velocity measure. Micromanagers beware, keep your grubby fingers off a team's user stories, story points, and velocity measures, if teams choose to use such practices, since they are not part of the Scrum framework. Even higher-level constructs such as visions, strategies in the form of lean canvases, and business objectives in the form of objectives and key results (OKRs) should be co-created with the lean-agile teams themselves from the bottoms up further promulgating the value and principle of decentralized decision making. That is, the lean-agile teams themselves may help select the market or market segment to pursue, help select the OKRs to determine market penetration, and then self-select the Sprint goals, user stories, tasks, and velocity necessary to achieve the OKR targets. Lean-agile teams will form business hypotheses, small business experiments, and rapid deployments to collect market feedback, compare as-is vs. to-be OKR performance and maybe even co-evolve the OKRs and solution sets together. Emergence is key within the lean-agile thinking paradigm and OKRs may emerge and evolve throughout the business experimentation process along with acceptable performance thresholds. For larger scale lean-agile frameworks such as SAFe, there may be a minimum set of recommended practices such as lean-agile product management and lean UX practices, program increment (PI) planning, scrum of scrums, system demos, inspect and adapt, and minimalistic measures such as program predictability. You're simply not doing SAFe if you're missing one if these key elements. It is often said that one isn't doing SAFe if you're not doing PI planning, but that is simply a cop out if you're missing lean-agile product management, inspect and adapt, and program predictability now. Therefore, lean-agile frameworks such as Scrum and SAFe create a minimum set of guardrails to help keep lean-agile teams on-track. And remember, the goal is not to turn lean-agile frameworks into feature factories for realizing over scoped integrated master schedules (IMSs), business requirements, enterprise architectures, and long lead technologies. Conversely, the goal is to innovate with small batches and limited WIP.

5. Em·pow·er·ment • O·ri·ent·ed (*ĕm-pou'ər-mənt • ôr'ē-ənt'ĭd*) Allow, grant, enable, license, warrant; To grant equal decision-making rights, powers, and authority for selecting, improving, and innovating policies, procedures, and practices

- ✓ Self-organizing teams.
- ✓ Loosely structured guardrails.
- ✓ Confident, bottoms up behaviors.

A very closely interrelated principle of SAFe DX is empowerment-oriented teams and individuals. The goal of decentralized decision-making and empowerment-oriented teams is to push the power, innovation potential, and as many decisions to the edge as possible. When lean-agile teams are empowered, then decision making speed is increased, business experiments are conducted faster, innovative solutions are created quickly, market feedback is rapidly collected, and the teams can rinse and repeat as many times as are necessary to converge upon a viable innovation in the shortest possible lead and cycle time. Put up roadblocks like centralized decision making; engineering, architecture, or configuration control boards; stage gates, gate reviews, milestone reviews, security reviews, and other disempowering controls; and lead and cycle times become enormously large, and innovation is stifled. The butterfly effect is in play here and a single disempowering gate review can cause a lean-agile team to miss a valuable market window, opportunity, measurement point, and revenue stream (much less dozens of such disempowering steps). Place the decision-making power in the hands of the lean-agile teams themselves, automate as many control points as possible, and get the engineering, architecture, or configuration control boards out of the way, and watch the innovation potential of your teams flourish. Lean-agile teams must be empowered to form their own hypothesis, create solutions to evaluate those hypotheses, and deploy them as many times as possible directly to the production fabric of your enterprise in order to collect valid measures and rinse-and-repeat as many times as possible. Of course, there must be some guardrails in place to govern the types, kinds, and classes of hypotheses, experiments, solutions, and measurements that can be collected. Lean-agile teams are exactly that, teams, so accountability is ensured when teams are making group decisions to prevent individuals from making rogue decisions. Lean UX guardrails and architectural runways may be in-place to help govern the overall direction of the teams to keep them from veering too far off in the wrong direction. Likewise, if a lean-agile canvas replete with strategic boundaries and OKRs is in place, this will also help keep lean-agile teams on track as well. With authority comes responsibility and teams are expected to review their results, outcomes, and tactical direction with lean-agile product management as well as portfolio management. This is not to say that portfolio nor product management are the exclusive owners of the lean-agile canvas and that the lean-agile teams themselves are not empowered to co-evolve the lean-agile canvas based on the results of their individual business experiments. With the right balance of lean-agile canvases, product management, lean UX, architectural runway, and empowered lean-agile teams, the portfolio, large solution, or small product and service ecosystem will experience unprecedented growth and innovation.

6. In·no·va·tion • O·ri·ent·ed (*ĭn'ə-vā'shən • ôr'ē-ənt'ĭd*) New, fresh, novel, modern, original; To establish an operating model, culture, and system for creating innovatively new delivery frameworks, products, services, and service products

- ✓ Out-of-the-box thinking.
- ✓ Business experiment oriented.
- ✓ Customer focused problem solving.

A critically important principle of SAFe DX is obviously the notion of creating innovatively new product, service, and service products. Innovation and innovative outcomes have many sources. First and foremost is listening to the voice of the customer, which means actually talking to your customers, identifying their pain points and goals, many of which exist as hidden, tacit, and inexpressible needs, and quickly closing their gaps and identifying solutions to address those pain points. Much of this involves going well beyond incrementally improving the efficiency of a single point solution, attribute, and feature, to creating entirely new ecosystems of previously unthought of solutions. This may be looking across the entire operational value stream, user experience map, or simple story or activity map to streamline steps or stages with long lead and cycle times, eliminate pain points and performance gaps, or create unprecedented conveniences. For instance, instead of just following Conway's Law and automating a laborious, multi-step, multi-transaction bureaucratic business workflow, why not reduce it to a single step or eliminate it all together. Take for example, Amazon's "one-click" checkout, painlessly requesting an Uber driver, or adding your favorite song to your smartphone's playlist quickly and easily. So, in retrospect, real customers or end users must be observed or interviewed, some lean UX applied, hypotheses formed, small business experiments solutioned, and measurable feedback gathered as quickly as possible, and this cycle has to be repeated a few times to get it right. It helps if the batch size is extremely small, the innovation cycle is short (i.e., a few hours, days, or week at the most), there is enough time to think, the innovators are talented, teamwork is in play to serve as self-governance, a malleable lean-agile technology fabric is used, and, of course, the team is not just a feature factory for one-and-done, fire-and-forget user stories. Again, it helps if bleeding-edge technologies are in-play, rather than just polishing last decade's rock with incremental efficiencies, or the team is creating entirely new technologies. Convergence is also a key, where multiple supply and value stream stages are merged, converged, and intermediated to reduce or eliminate customer and end-user pain points, inefficiency, inconvenience, and cost. Think of how many value stream or supply chain steps a single Apple iPhone subsumes (i.e., voice, email, GPS, texts, music, high-resolution photographic and videography, teleconferencing, web browsing, social networking, online gaming, entertainment, movie streaming, travel management, transportation management, online shopping, education, bill paying, etc.). A single Apple iPhone converged, eliminated, or replaced entire industries and brought multiple value stream stages together in a few convenient low-cost painless steps and exemplifies the notion of user experience design. A lean-agile team exists to innovate through superior user experience design vs. serve as a mindless velocity-driven feature factory.

7. Keep • It • Sim·ple (*kēp • 'ət • sĭm'pəl*) Easy, plain, basic, elementary, uncomplicated; To establish a basic, easy-to-use system for creating innovatively new products and services, along with the scale, scope, and size of the solutions themselves

- ✓ Simplest/smallest possible backlogs.
- ✓ Minimum viable product (MVP) focused.

✓ Extremely small batchsize and complexity.

An overriding or implied principle of SAFe DX is the imperative to keep-it-simple, which means exactly what it says, "keep-it-simple" and resist the temptation to make things too complex! This is an all-encompassing imperative that includes an enterprise's operating values and principles, lean-agile framework, governance structures, methods and practices, measures, and tools, and of course the scope of the products and services themselves. Humans naturally want to make things more complex than they have to be and "the-more-the-merrier" seems to be the mantra of traditional enterprises. We want more values, principles, frameworks, governance boards, methods, practices, tools, technologies, teams, people, etc. Besides, now with SCALED agile frameworks, there's no sense in limiting the scale, scope, and magnitude of the products and services anymore, the gloves are off, and we can build 300, 500, and even 1,000 person teams. First, keep your values and principles small, and, yes, select an appropriately sized agile framework (SAFe is a good choice), but, beyond that, use the fewest number of teams, smallest team size, and simplest product and service scope possible. This also includes the lean canvas, market segment, objectives or OKRs, lean-agile product management, lean UX, and individual team product backlogs. Once again, humans have rather large brains, they are quite energetic and imaginative, some are simply hormonally imbalanced, and they can dream up inordinate amounts of complexity. OKR maps can easily exceed dozens if not hundreds of measures, portfolios contain dozens of initiatives, product ecosystems contain hundreds of people, frameworks contain hundreds of processes, documents, and measures, and decision points easily reach into the hundreds and thousands. Whatever happened to the butterfly effect? It's more like the Godzilla effect? How many mutant reptiles does it take to consume the planet in one bite? Don't be a Godzilla and keep-it-simple. A simple narrowly scoped lean-agile canvas will suffice, an OKR or two is enough, a simple lean-agile framework is more than enough (it ain't rocket science), two or three teams will do just fine, a little lean UX is okay, and a few small business experiments are more than enough to scale up and change the world. We have to get out of the mindset that 500 people, a billion dollars, a decade long integrated master schedule (IMS), a stack of business requirements, and multi-million-dollar enterprise architecture is required to make a dent on the marketplace! Facebook was originally created by one programmer in his spare time and has a $1 trillion total market capitalization. Geez, how many lean-agile teams, user stories, story points, and velocity measures do you think you need to top that! Inordinately large lean-agile portfolios, solutions, and product teams simply consume resources, they do not generate revenues. Keeping the lean-agile canvas, OKRs, lean-agile framework, product scope, and number of teams simple is what generates revenue!

8. Make • It • Pain·less (*māk • ′ət • pān′lĭs*) Fast, easy, effortless, trouble-free, frictionless; To establish a lean-agile operating model, delivery process, and solution set that is not overly burdensome, complicated, nor bureaucratic to apply

✓ Simplest possible day-to-day activities.
✓ Simplest possible lean-agile framework.
✓ Simplest possible product-service scope.

Similarly, a critically important principle of SAFe DX is to make-it-painless. This includes every element of SAFe from top-to-bottom (i.e., portfolio, large solution, essential or program-level). Once again, there's a tendency to overwhelm the SAFe system with too much WIP, after all, SCALED is in its name (i.e., too many epics, capabilities, features, user stories, tasks, metrics, roles, teams, individuals, ceremonies, etc.). SAFe is a little like sex, if it is painful, you're probably doing something wrong, that's the bottom line! Some of the things that make SAFe painful are skipping essential steps like value stream mapping; skipping portfolio, large solution, and ART canvases; or simply overwhelming portfolio, large solution, and program backlogs with 15-year-old business requirements documents. First of all, a business requirements document is neither an epic, capability, feature, nor user story, so simply stop it! Forming inordinately large or ill-formed solution and agile release trains is another common malady. Having no portfolio, solution, or product management team is a common anti-pattern (i.e., who needs them if there are a stack of 15-year-old business requirements documents laying around). Forgetting about lean thinking and lean UX and building over scoped, gold fleeced, and gold-plated systems is also a major problem in SAFe. Failing to train and certify personnel, nor follow the SAFe implementation roadmap is clearly a major problem. Assigning exactly one portfolio, solution, or product manager, or one solution or release train engineer is a major problem. The span of control for one RTE is instantly violated, which should really be an RTE team. Of course, a lean-agile center of excellence (LACE) nor agile program management office (APMO) are rarely formed, leaving all lean-agile governance to the already overwhelmed RTE, who must also be the product manager or system architect too in most cases. Full-time SAFe coaches are a luxury, rarely staffed, or relegated to part-time training or scrummastering roles instead of SAFe performance managers. There are far too many product owners and scrummasters, and most teams are functionally oriented increasing the number of dependencies between teams beyond what is manageable. SAFe and Scrum ceremonies become torturous because of overwhelming backlogs, and RTEs want multiple scrum of scrums per week overwhelming scrummasters and the teams themselves with too many reporting responsibilities. Of course, without solution nor product management and sufficiently groomed backlogs, program increment (PI) planning becomes a boondoggle, eight-hour days become 12-hour days, planning has to be stretched to three or four days, teams continue planning for two or three weeks, and RTEs complain that iterations plans diverge from PI plans. It all boils down to too much pain, very little benefit and payoff, and a lot of demoralization. Everything can be automatically relieved by lean thinking, keeping-it-simple, and resisting an IMS-driven feature factory.

9. Make • It • Fun (*māk • ′ət • fŭn*) Playful, pleasant, gamified, entertaining, recreational; To establish an operating culture, lean-agile system, and set of innovatively new products and services that are engaging, pleasant, and enjoyable to apply

✓ Gamified lean-agile framework.
✓ Enjoyable vs. high-pressure environment.
✓ Simple, but challenging product-service scope.

A closely interrelated principle of SAFe DX is to make-it-fun. Yes, lean-agile values, principles, frameworks, methods, practices, tools, measures, and technologies can be fun! SAFe, much like Scrum, is a simple, slim-downed, and highly gamified lean-agile framework. Serious manufacturing, mathematical, and Kanban mechanics have a hard time seeing the hardcore science in SAFe—That's because there is no hardcore science in SAFe. If you're looking for calculus, physics, chemistry, or manufacturing principles, pick up a college textbook on one of these topics—You aint' gonna find it in SAFe! This isn't to say SAFe isn't based on science—Instead, SAFe is based on a few simplified concepts. These include simple value stream mapping, lean business model canvases, elementary Kanban practices, etc. SAFe comes replete with pseudo-scientific lean-agile principles such as economic thinking, visualization, limit WIP, small batches, decentralized decision making, etc. This isn't to imply there are no quantitative lean-agile practices in SAFe, because there are (i.e., WSJF, WIP limits, cumulative flow diagrams, etc.). However, the goal is innovation, the approach is to keep-it-simple, and you're definitely gonna break SAFe if you turn it into an integrated master schedule (IMS) or business requirements driven feature factory with full-utilization, split second tasking, earned value management (EVM), critical path management, and manufacturing statistics. SAFe will break your IMS in a microsecond, over and over again, and all-of-the-king's-horses-and-all-of-the-king's-men-can't-put-Humpty-IMS-together-again! Conversely, you can strive to make SAFe fun, enjoyable, and worthwhile. Instead of turning your solution or agile release train into a bunch of SAFe haters with overwhelming backlogs, never-ending SAFe and Scrum ceremonies, and mindless story point counting at the highest levels (for which story points were never intended), strive to attract vs. repel your lean-agile teams. Make value stream mapping and lean canvasing fun, simple, electronic, interactive, rewarding, and gamified. Keep the number of epics, features, and user stories to the bare minimum number possible. Build in plenty of excess capacity into program increments and sprints. No self-respecting data center operator would load a microprocessor to 80% capacity, so why would you think you could load a human innovator to 80% capacity? Co-form real business experiments, keep them infinitesimally small, build in plenty of innovation time (i.e., if the WIP is low enough, a week or two is plenty of time for a small business experiment), and rapidly deploy business experiments to your enterprise's production fabric for direct exposure to customers (i.e., that's what Google, Apple, Amazon, Facebook, and Microsoft do). There is nothing more enjoyable and rewarding than empowering front-line innovators to deploy business experiments to end users and instantly collect measurable feedback (i.e., sans the architecture, engineering, or configuration control boards).

10. Small • Batch • Size (*smôl • băch • sīz*) Set, lot, pack, bunch, group, collection; To create a set of innovatively new products and services that are small, simple, uncomplicated, quick, and easy to deploy without a great deal of effort, pain, or burden

✓ Smallest possible product-service scope.
✓ Smallest possible lean-agile footprint.
✓ Fastest possible delivery cycle.

One of the most important principles of SAFe DX is small batch size. It is the foundation upon which lean thinking, innovation, and DX are built. That is, keeping the scale, scope, and magnitude of enterprises, portfolios, solutions, product ecosystems, products, MVPs, business experiments, numbers of teams, and lean-agile framework as small as possible. Small batch size creates a cascading waterfall of beneficial effects. For one, small batch size improves workflow as it is easier to pull a small batch through a workflow faster than a larger one. Think of a single user story or work item as the smallest possible batch size for a moment. A user story is NOT a testable systems requirement nor mathematical equation, but an agreement to have a conversation. For instance, an example of a user story may be, "As a project manager, I would like a query capability to return a list of user stories so that I may determine the number of open user stories." Okay, simple enough, but English is a natural language with full of ambiguities. A developer may ask the product owner whether this is an SQL query? What database and version are begin used? How big is the database? How many user stories can it return? What is the format of the results? Is this in the form of a list or a graph and what kind of a graph? Are they open, closed, or pending user stories? Should the query support other determinants such as team, project, date ranges, overdue user stories, invalidated user stories, upcoming user stories, open user stories, etc.? The number of potential questions generated by this single user story may be infinite and the product owner may have mis-specified the original story and really wanted a pie chart with open defects or realized it was the number of planned vs. actual story points that were needed? Now, make the classical mistake of paying a big six consulting firm several million dollars for a few thousand user stories! Multiply the number of user stories by the number of interpretations and questions each generates. Let's say, the big six consultant generated 5,000 user stories and the number of possibilities for each is about 25, just to be conservative. You've now generated 75,000 possible variations, maybe none of which are actually needed. Let's say the average story takes about 3 person days or about $5,000 per user story. Your business requirements specification is now worth $375 million. That's about 15,000 Scrum sprints or about 288 staff years if you average 5 good size user stories per sprint. Of course, the number of conversations and communication paths would be infinite, you would instantly freeze the queues and the actual effort may be more like 2,000 staff years if you cut the number of requirements in half and don't care about technological obsolescence, quality, or customer satisfaction. How about we focus on a small number of business experiments (i.e., five or ten), quickly deliver them and gather customer feedback, and try to get those right? The economics of keeping-it-simple far outweigh traditional IMS-driven fully utilized feature factories.

11. Small • Ex·per·i·ments (*smôl • ĭk-spĕr'ə-məntz*) Test, trial, probe, model, sample, prototype; To create innovatively new products and services as small hypothesis tests designed to gather rapid measurable feedback and dispel uncertainty

✓ Lean startup lifecycles.
✓ Small, fast business experiments.
✓ Fast measurable feedback/improvement.

A similarly important principle of SAFe DX is to keep the size of the experiment as simple as possible. Let's say for instance, you have solid lean agile leadership and they've decided to trash their 15-year-old stack of business requirements which a very wise choice in lieu of a few business experiments. This is quite the progressive leader, indeed! However, each business experiment is epic or feature sized, completely over-scoped, and will certainly take a few decades, years, or program increments to complete. This is not an entirely uncommon scenario. In one case, leadership decided to build a on-premises cloud in a highly regulated environment with process and document driven waterfall lifecycles, port 70 obsolete legacy systems to the cloud to query their data out at one time while preserving their original state, and implementing a shiny new agile lifecycle management system for timekeeping purposes. While that may seem like three epic-MVPs, that was really several volumes of business requirements masquerading as a few simple business experiments. Needless to say, those business experiments cost nearly a billion dollars and a decade, while the technological clock was ticking (i.e., with each passing day, the amount of technical debt was accumulating exponentially). It's also pretty common for UX teams to create epic or feature sized user stories, cram 5 to 10 of these feature stories into a two-week sprint, and then wonder why it takes 6 months to finish the sprint. The far better choice is to create SMALL bite-sized business experiments (i.e., improve system performance by 100 milliseconds, limit the size of an input box on a free form comment field, consolidate a few choices into a one-click function, simplify menu choices, or some other incremental improvement). A small business experiment such as this can be solutioned by a small team pretty quickly, deployed to an enterprise's production fabric, and the results can be measured in a matter of hours. If the new function enhances system performance, number of transactions, number of help desk tickets, or even revenues or profits, then the new function can be validated as quickly as it is deployed. So, yes, dump your integrated master schedules (IMSs), business requirements, and enterprise architectures, have your UX teams produce a few possible wireframes, and apportion some of their possible attributes into a few small business experiments. However, don't go overboard, create hundreds of epic-sized wireframes, gaslight yourself and your Scrum team into believing the epic is a user story, and program dozens of epic-sized wireframes into a few sprints. Instead, create a forward thinking UX wireframe, decompose or identify a few salient characteristics, and allow innovation teams to pull these through the business experiment Kanban one attribute at a time, create a solution, gather customer feedback, and rinse and repeat few times before pulling the next attribute into the queue. A small business experiment is exactly that, SMALL (vs. epics masquerading as user stories).

12. Small • De·ploy·ments (*smôl • dĭ-ploi'məntz*) Send, ship, deliver, launch, release, install, distribute; To develop innovatively new products and services as small business experiments that can be rapidly and frequently delivered early and often

- ✓ Rapid daily production deliveries.
- ✓ Fast measurable market feedback.
- ✓ Hypothesis-driven customer testing.

Another critically important principle of SAFe DX is the application of small deployments. The notion of deploying a small business experiment to the enterprise's production fabric goes without saying and is a little more complex than it sounds. Many "feature factories" are more than satisfied to push user stories into a queue for validation, staging and certification area, or other large batch holding bin. It doesn't take a small set of software teams very long to create a few hundred thousand lines of Java. Traditionally, the Java is kept in a single version control repository (if you're lucky) and someone tries to compile, build, and test the large batch of Java code all at one time six or nine months after it is created. IF you can get the batch to compile and build at all, then, of course, someone has to test it. It usually takes a small army of independent testers a few weeks to test this large batch of code. Of course, performance of the end-to-end code has not even been considered, hundreds of priority one defects instantly emerge, not including lower priority annoying defects, and, of course, system reliability, availability, stability, or security is a non-starter. Some organizations devote tens of thousands of hours to such events. Let's say a tester is half the cost of a software developer, so that's still about $2 million for the testing event, testers are burnt out go postal and quit, and all you have is a bunch of unworkable code, hundreds of priority one defects, a very angry customer, and a demoralized software development supplier team. Instead, keep the number of business experiments to the bare minimum, implying a very small code base, keep the business experiments themselves very small, implying an even smaller code base, and deploy the small business experiment directly to the production fabric using a DevSecOps pipeline. That is, a small team forms a hypothesis (i.e., reducing checkout speed by 100 milliseconds will result in more transactions, revenues, and profits), codify the hypothesis in a few lines of code (i.e., reworking the credit card processing algorithm), check it into version control, run a few thousand automated tests (in seconds), and deploy the new function directly to the market. Measure the number of transactions before, during, and after the deployment, determine if their change can be attributed to the new code, and even monetize the transactions to determine revenues and profitability. A few statistical tests may be in order to determine cause and effect. If the new deployment degraded system performance, revenues, or profits, or there was no noticeable change in results, then the next choice may be to roll back the change (or continue refining the hypothesis to achieve the desired results). There are several concepts in play here (i.e., small number of experiments, small size of experiments, and deployment directly to production fabric via a DevSecOps pipeline). There are a few other concepts like hypothesis testing, statistical analysis, and monetization of the results. There is no batching for late big bang testing.

13. Small • Fea·ture • Teams (*smôl • fē'chər • tēmz*) Crew, group, squad, assembly, collective; To apply small teams of people with complementary technical skills, abilities, and talents that can form vertical slices of valuable market products and services

- ✓ Form small, cohesive teams.
- ✓ Ensure all skills are included in team.
- ✓ Ensure teams can deliver vertical features.

A tried-and-true principle of SAFe DX is a small cross-functional feature team, which is a small group of people with complimentary skills. Perhaps one person understands the network or operating system topology, another the middleware business process workflow engineer or backend database topology, and yet another the actual business logic or front-end user interface technology, and yet other the finer points of UX, testing, security, performance, and scalability concerns. The point is that a single team should have at its disposal the full gamut of technical skills necessary to design a vertical feature slice without the necessity to depend on disparate functional teams. Yes, the notion of complicated-subsystem, stream-aligned, platform, and enabling teams has resurfaced, but this obfuscates the necessity to focus on a small number of simple, but impactful business experiments that solves large problems with equally large business value in return. If the business experiment is small enough, then the team doesn't have to be very large. For instance, a small lean UX team of about six or seven people can design an entirely new smartphone app in about a week or so (and the same could be said for almost any domain using Google's five-day design sprint process). However, these are pretty complex epic or feature sized designs these teams are creating in only five business days (and an even smaller team of people with complementary skills can design an even smaller business experiment). It really only takes about a two-person pair programming team with complementary skills to refine a small business experiment in only a few hours, days, or weeks. However, the point is that the business experiment, whether it is an entirely new customer facing function or enhancement to the invisible underlying performance of an existing function, the team certainly doesn't have to be too large. Oftentimes, three people are a crowd and dissention immediately occur beyond about two people. Four to five people is manageable if the stars align, and anything beyond that gets a little unwieldy. Now, consider a small product ecosystem with multiple teams. While it is possible to align the work of multiple small teams under a single lean-agile product management or lean UX team or product owner, anything beyond about three to five teams becomes unwieldy as well. Some larger-scale lean-agile frameworks like SAFe are well adapted to synchronizing the work of multiple teams, but even with polished frameworks like SAFe herding cats can be a bit of a challenge in the Western hemisphere where fierce individualism prevails. Extremely small well-oiled teams or small groups of teams often outperform larger teams or larger groups of teams, and the larger your team or team of teams, the more likely that free riders, under performers, or wasteful non-productive teams exist. So, be very careful when your team of teams gets too large. More people doesn't mean higher productivity and the science of queuing theory illustrates that the bigger it is the harder it falls.

14. Small • Proj·ect • Size (*smôl • prŏj'ĕkt' • sīz*) Job, task, activity, assignment, undertaking; To form and apply the smallest number of people necessary to create a small, highly cohesive ecosystem of innovatively new products and services

- ✓ Small product-service scope.
- ✓ Smallest possible ecosystem team.
- ✓ Fewest number of feature teams possible.

A closely related principle of SAFe DX is the practice of small project size. That is, the smallest number of teams and people possible to create a small highly cohesive ecosystem of innovative products and services. This includes lean-agile thinking practices like rapidly deploying a small number of tightly scoped business experiments to your enterprise's production fabric to collect measurable feedback from customers. Although the new vernacular includes project-to-product thinking such as longer-lived development value streams vs. short-term traditional projects with a clear beginning and end. These are often referred to as a team of teams or agile release train (ART) in SAFe. The goal in SAFe is for the ART to behave as a single large-scale cross-functional team. Each team contributes a feature or set of user stories to a program or product backlog formed by a SINGLE lean-agile product management or lean UX team. This implies a lean-agile product management team exists, it is applying lean-agile UX, and it is devising a small set of extremely tightly scoped business experiments. More than one team exists in SAFe, but they are consistently applying uniform practices, tools, and measurements; synchronized and sprinting together in cadence; and contributing to a single larger solution. SAFe's ceremonies, like those of Scrum, are a forcing function for cooperation, collaboration, and consensus-oriented thinking typically found only in Far Eastern cultures where conformance to the greater good is hammered into the psyche of every man, woman, and child. This, of course, is very difficult to achieve in the Western hemisphere where fierce individualism has been programmed into every man, woman, and child for millennia. That is, each team will have varying sizes and composition, select their own practices and measures (including none at all), give lip service to lean-agile practices, refuse to sprint together in unison, race against one another for best velocity measures or innovative solutions, even game velocity measures to compete with one another, or RTEs pit them against one another. It's difficult to achieve SAFe's values of alignment, synchronization, cadence, and consistency when cats run in entirely different directions. The cats will even undermine the credibility of their own team members to feed their own insatiable egos. Now, scale this up to 10, 20, 30, or even more teams than that, and you now have 300 to 500 cats clawing at one another for scarce resources, attention, self-promotion, and cannibalization of each other's reputations. Furthermore, SAFe suggests a single STE or RTE can manage dozens of teams and hundreds of cats, and now you have a single autocratic leader trying to operate a feature factory in a vacuum, without the aid of a LACE, APMO, and SAFe coaches (i.e., power corrupts, but absolute power corrupts absolutely). What's the bottom line? Yes, use SAFe but scale your ART down to something more manageable and consistent with small teams at every level (i.e., including small STE and RTE teams).

15. Small • Prod·uct • Size (*smôl • prŏd'əkt • sīz*) Tiny, little, minute, petite, miniscule; To create the smallest possible ecosystem of innovatively new products and services that provide the greatest possible market impact

- ✓ Small product-service ecosystem.
- ✓ Smallest possible ecosystem structure.
- ✓ Fewest number of product features possible.

Another important principle of SAFe DX is small overall product size. There are manageably small enterprises, portfolios, solutions, lean-agile frameworks, teams of teams, individual teams, business experiments, batch sizes, and deployments. However, all of these are just a means to an end, which are the smallest possible product sizes. A series of extremely small business experiments may amalgamate, aggregate, or evolve into a large product and service ecosystem, in fact, they should. However, in the end, the scope of the final product should be kept as small as possible too. Think of a small smartphone app like Google Maps. First of all, the interface may seem small, but it is anything but simple. Imagine the complexity of the smart phone ecosystem in which the GPS resides, much less the large and complex cell phone network infrastructure necessary to serve up the data and keep your automobile on course (accurately). And, of course, imagine the back end where the maps and points of interest are served up, which itself a modern technological network (i.e., massively parallel processing, multi-petabyte scale, high performance cluster serving up billions if not trillions of requests in fractions of a second). Yes, indeed, Google Maps is the perfect 21st century app, almost a microservice, with an impactful footprint with hundreds of millions of end users serving up trillions of transactions a day without skipping a beat. In spite of its enormous complexity, it's really quite simple and exemplifies or embodies the principle of small product size. It's a standalone app, with a few functions and features, and few unnecessary bells and whistles. As the old saying goes, "Any intelligent fool can make things bigger and more complex but takes a touch of genius and a lot of courage to move in the opposite direction!" Any reasonable traditional product manager would certainly create a mountain of business requirements, hire a thousand people, and run a billion-dollar project to realize such a backlog. And that's not far from the truth, except that Google Maps was originally coded by two people and sold to Google who evolved it into a billion-dollar project over a decade from 2005 to 2015, serving over one billion customers. The point is that Google Maps is a simple application, it was created by two people, and it was evolved into a global success through a series of small and relatively inexpensive business experiments. Yes, Google had to build a family of massively parallel processing high performance clusters to run it, but even those aren't terribly expensive nor difficult to build (i.e., about $10 million apiece or less and fabricated in 90 days or less). Furthermore, most of the back-end software was reused from other projects like its search engine. The most expensive and labor-intensive part, in addition to a plethora of acquisitions in order to assemble its ecosystem, was certainly the maps. Google set out to create its own maps instead of paying exorbitant licensing fees, patents, and copyright fees to existing map makers. Yes, expensive, but functionally simple.

16. Lean • Prod·uct • Man·age·ment (*lēn • prŏd'əkt • măn'ĭj-mənt*) Goods, produce, commodity, component, merchandise; To develop and apply a formal upfront process for scoping extremely small batches innovatively new products and services

- ✓ Formal product management process.
- ✓ Formal product management team.
- ✓ Lean-agile product thinking.

Yet another critical principle to SAFe DX is the creation, operation, and application of a lean product management team. The art, science, and practice of traditional, lean, and agile product management is still hit-and-miss, inconsistent, and mostly non-existent. A lean product management team is just that, it's a team of people who are responsible for identifying, scoping, and planning a product or product ecosystem. It's a confluence of marketing, sometimes systems architecture, and project management in many regards. One or more individuals must perform market analyses, develop a lean canvas and business case, get executive stakeholder buy in and funding, document its requirements, and form and sometimes manage a product development or engineering team. That's really a small army of people, which is almost true of any lean-agile function but is often relegated to a single overworked individual. When product managers are appointed, they are usually handed a product scope and simply told to realize it without having any say in its evolution. So, traditional product managers are between a rock and a hard place (i.e., non-existent, overworked, or patsies for the projects of executive pipedreams). Enter lean product management, which is, yes to apply some principles and practices of traditional product management, of which there are multiple emerging bodies of knowledge, but more importantly apply lean thinking principles and practices. Lean product management teams still do market analysis and segmentation, business model generation in the form of lean canvases, develop business cases, get budgets, and form and manage product development or engineering teams. The difference, of course, is that lean product management teams are working faster, narrowing down the scope to a few salient hypotheses, and schedule a series of extremely tightly scoped business experiments in sparsely populated near term product roadmaps vs. long-term EVM-driven integrated master schedules (IMSs). Lean product management embraces lean thinking principles, build in excess capacity, and empower lean-agile teams to rapidly field a series of small business experiments to capture market feedback. Of course, lean product teams are evaluating this feedback too, evolving their roadmaps, adjusting and redirecting experiments, and leading lean-agile teams to a point of convergence. The currency of lean product management is visions, canvases, a few salient OKRs and leading indicators, market feedback, and rapidly evolving sparsely populated near term roadmaps with tons of white space visually representing high levels of uncertainty associated with the very act of innovation. A densely populated IMS, business requirements specification, or enterprise architecture neither represents lean thinking, uncertainty, nor innovation. When can lean-agile teams innovate with suffocating and densely packed user stories representing full utilization and no room to breathe? There is no innovation when people are underwater suffocating to death!

17. Lean • Us·er • Ex·pe·ri·ence (*lēn • yo͞o'zər • ĭk-spîr'ē-əns*) Useful, helpful, valuable, practical, effective; To apply a simple set of user experience design techniques for creating innovatively new products and services that customers need and want

- ✓ Customer-market innovation focused.
- ✓ Small, fast business experiments.
- ✓ User experience design thinking.

A closely interrelated principle of SAFe DX is the application of lean user experience or lean UX practices by a lean product management team. So, this is a two-fer, two principles and practices for the price of one. You've got to have lean product management team which most lean-agile projects do not have, and more importantly, the lean product management team should be applying lean UX practices. Much of the discipline of lean UX is encapsulated by the fields of user centered design (UCD) and design thinking principles and practices, which establish the traditional foundation of lean UX. The difference is that lean UX teams apply lean thinking principles (i.e., they slim down the number of UCD and design thinking principles into the converged discipline of lean UX). Lean UX is composed of value stream maps, lean canvases, value proposition maps, lean roadmaps, impact maps, user experience maps, such as empathy maps, journey maps, personas, UX wireframes, story maps, and variety of interrelated tools such as innovation games and techniques such as jobs to be done. Lean UX tools have many things in common, i.e., they are simple, visual, analytical, and replace multi-year marketing analyses and business plans, integrated master schedules (IMSs), business requirements, and enterprise architectures. There is even some business case analysis, economic thinking, and prioritization-based tools such was weighted shortest job first (WSJF). The point is to keep-it-simple, apply a few meaningful lean UX tools, limit batch sizes, limit WIP, and reduce utilization. It's easy to go overboard with these tools, design densely populated 500-page PowerPoint briefs in 8-point font, or develop a mountain of UX wireframes, or overpopulated story maps. Yes, some brainstorming and analysis is required and may result in overpopulated lean UX visualizations, but the key is to down select, prune, and strip them down to a sparsely populated MVP that can be translated into a small number of extremely simple and tightly scoped business experiments. That is where the lean thinking enters the lean product management and lean UX equation (i.e., keeping it simple, sparsely populated, and emergent). Lean business experimentation is not a one-and-done, fire-and-forget activity, it is rather a rinse-and-repeat activity. So, lean product management teams should not jam product backlogs chock full of over-scoped epic and feature sized user stories nor attempt to squeeze every penny of labor out of lean-agile teams with full-utilization. Lean product management teams are also lean-agile teams, they follow lean-agile practices such as Scrum or Scrumban, they are accountable to a Scrum of Scrums, and they should be on the ART in the case of SAFe. They should exist, they are not rogue teams, they are not street cats that cannot or should not be herded, they are responsible for participating in events such as sprint and system demonstrations, and they have their lean UX backlogs groomed well in advance of SAFe PI planning.

18. Lean • De·sign • Sprints (*lēn • dĭ-zīn' • sprĭntz*) Span, time, period, interval, duration; To quickly scope innovatively new products and services customers need and want in small one or two week timeboxes leaning toward the shorter timescale

- ✓ Highly-structured design process.
- ✓ Fast, tightly-scoped product cycles.
- ✓ Fast market feedback improvement.

An emerging principle of SAFe DX is the practice of lean design sprints. These are simply early discovery sprints used to scope out the initial design of a product or service. They are short lean business experiments, typically conducted by business executives such as the CEO, marketing, product managers, project managers, designers, developers, and customer-facing experts. Design sprints are a five-day, five step process of mapping out the problem space, sketching competing solutions, making difficult decisions, hammering out a prototype, and demonstrating them to real customers. The first step is kin to a mini lean canvas, the second step is like an alternatives-of-analysis (AoA), the third step consists of selecting the best idea and merging ideas from other alternatives, the fourth step consists of prototyping the final design, and the fifth step consists of allowing actual customers or end-users to evaluate your prototype. Design sprints are a confluence and mashup of lean thinking, strategic planning, marketing, product management, system design, agile methods, user centered design (UCD), design thinking, lean UX, rapid prototyping, and, of course, rapid small business experiments that are released to a high-fidelity production fabric (i.e., live end user). Many prototypes produced by design sprints are released into the wild (i.e., show cased in live settings whereas the end-user does not even realize its an advanced prototype, but thinks they are reacting and responding to a production unit). In the case of web or application design sprints, they can be conducted in a customer setting, live customers can be included in the process, designers may construct a real-time story map, developers may prototype live code, and multiple revisions of the prototype may be shown to live or in vivo customers for fast feedback. Once again, design sprints exhibit many properties of lean thinking (i.e., they are collaborative and involve teamwork, they are timeboxed, they are based on lean UX artifacts, feedback is sought each and every day, a tightly-scoped near production prototype is produced, and the goal is live customer feedback—Did we get a thumbs up or thumbs down). Based on the feedback, executives, product managers, and other key decision-making executives will form and fund the epic-MVP, a project or product development team will be formed, a lean product management team will groom a backlog, and innovators will produce a customer-validated production unit in short order (also as quickly as possible). Of course, leading metrics will be applied to gauge market, customer, and end-user validation, and used to make pivot or persevere decisions. This process is superior to hiring a big six consultant to fabricate a mountain of business requirements, integrated master schedule (IMS), or enterprise architecture, and program these assumptions into a 5-10-or-15-year multibillion-dollar project with little market impact. Design sprints help scale and deliver narrowly scoped MVPs to large numbers of end users with impact measures.

19. De·sign • for • Scale (*dĭ-zīn' • fôr • skāl*) Extend, expand, enlarge, magnify, lengthen; To design and develop innovatively new products and services for the largest number of simultaneous users possible for maximum business and market impact

- ✓ High-performance oriented design.
- ✓ Global market scale design thinking.
- ✓ Design for greatest possible market impact.

An elemental principle of SAFe DX is design for scale. Oftentimes throughout the 20th century, innovatively new products and services were directly designed for a few lead users, a few hundred at the most, or a couple of thousand end-users in the worst case. Think of the mainframe, personal computer, or even digital camera, as their inventors never imagined more than a few units of sale a the best, and they were priced accordingly. That is, the entire research and development cost was transferred to the first dozen units or so. So, a billion-dollar R&D project transferred its costs in the form of a $100 million price tag for the first ten or so units. However, in each of these cases, these predictions were completely unfounded as millions of mainframes, personal computers, and digital cameras were eventually sold. Think of a modern smartphone, with the power and functionality of a mainframe, personal computer, and digital camera combined—Today, there are over 5.3 billion smartphones in use. Now, that's scale! Think of how many mobile apps a modern smartphone has—Perhaps a few hundred—That's over one trillion mobile apps! Today's innovators no longer have to think about spending one billion dollars in R&D to develop a mobile app and charge the first 10 customers $100 million each. Instead, a small team of innovators can design a new mobile app in a week for well under $200,000 and transfer the costs to a billion end-users. That's not even a single penny per end user! So, how exactly do modern mobile phone app developers make money? Sometimes it is in the form of advertisements (i.e., Google Search, Google Maps, and Google Ads sell business rankings to advertisers). In many cases, the customer or demographic data alone is worth billions of dollars (i.e., who is buying what and when, what data people are searching for, what preferences people have, etc.). More importantly, in today's global marketplace, millions of innovators are competing for the same customer base (i.e., you are not the only one developing a particular type and kind of mobile app, so you have a lot of intense market competition from people with high levels of motivation, low production costs, and better intelligence than you do). Therefore, the modern currency is speed, innovation, and validation! It is certainly not integrated master scheduling (IMS), business requirements analysis, enterprise architecture, or maturity model compliance skills that require a billion dollars and a decade to execute. Instead, today's innovators must identify a plausible solution, develop a simple low-cost MVP, and deploy it to an enterprise's production fabric in about a week for immediate validation. Winner takes all, and those who can apply modern lean-agile thinking principles garner revenues, sales, and profits from billions of global end-users. Individual unit pricing almost never enters into the modern equation, because instant market penetration to billions of end users is your currency to sell to the highest advertising bidder, which is not a bad windfall for a small investment.

20. De·sign • for • Test (*dĭ-zīn'* • *fôr* • *tĕst*) Assess, verify, validate, evaluate, appraise; To design and develop innovatively new products and services that can be tested early, continuously, and automatically throughout their operational lifecycle

- ✓ Test-first design principles.
- ✓ Small, fully tested deployments.
- ✓ Continuous automated system tests.

A closely interrelated principle of SAFe DX is the practice of design for test. That is, if you're going to design for scale to the tune of a billion global end-users in the form of a lean business experiment in about a week, then you better test it. This doesn't mean statistical hypothesis testing, which is a very important final step of a lean business experiment but verifying and validating the lean business experiment as it is being deployed to an enterprise's production fabric without an additional penalty of time. This of course implies that the innovative product or solution in the form of a lean business experiment is designed to be tested (i.e., it has acceptance criteria, automated test cases, and all automated tests must pass "before" it hits the production fabric and is released to your market, customers, and end users). For instance, 50,000 to 60,000 Google software developers commit up to 75,000 deployments directly to its production fabric each day in the form of small rapid lean business experiments resulting in over 200 million automated tests against those deployments before it reaches production platforms. On average, each small individual code deployment results in 3,000 automated tests run in fractions of a second. If a single test fails, the commit is rejected until all tests pass. This motivates small lean-agile teams of innovators to write clean code from the get-go that will pass its automated tests. The goal of course, is not to pass the tests, but to reach Google's production fabric, expose the lean experiments to production platforms, and collect qualitative and quantitative data from millions of live users. The purpose of each lean experiment may be to add new, but minimally essential functionality, enhance an existing function, simplify or consolidate one or more functions, improve their performance, reliability, or security, and improve overall end-to-end user experience. Google applications are some of the most spartan in the history of the world, they are free of excess dross, waste, work-in-process, gold plating, gold fleecing, and other unnecessary bells and whistles. In doing so, i.e., keeping their lean business experiments as small as possible, they increase each one's testability, actually test it, and ensure that all lean business experiments pass functional and non-functional tests "before" they reach its production fabric. Google has one to two billion end users or more, which conduct well over 40 billion transactions per day that means each transaction must work precisely as planned. Google itself pioneered the one-week design sprint to ensure the success and reduce the cost of failed traditional project management thinking, and increased its reliance on design for test, and not just for code, but outcome testing as well in form of statistical hypothesis tests. On average, nearly 95% of lean business experiments do not satisfy hypothesis tests, but yield priceless business intelligence. Failed experiments may be rolled back, new ones quickly deployed, more priceless data collected, and continue relentlessly to stay ahead of global competition.

21. De·sign • for • Se·cu·ri·ty (*dĭ-zīn'* • *fôr* • *sĭ-kyo͝or'ĭ-tē*) Guards, defenses, assurance, protections, precautions; To plan for and build in information, system, and application security principles when creating innovatively new products and services

- ✓ Application security lifecycles.
- ✓ Application security practices and tools.
- ✓ Application security technology, evaluation, and testing.

An often-forgotten principle of SAFe DX is the practice of design for security. It's just like it sounds, as information, information technology, and application security must be planned from cradle to grave, beginning to end, and concept to cash. This last term is one of the biggest reasons why as today's petabyte-scale information systems service billions of global end users in only a few short days, weeks, and months of lean business experimentation. That is, the goal of lean business experiments is to have the greatest global market impact possible. Doing so, enables the supplier of the innovatively new product or service to reach the broadest possible global market, therefore garnering the greatest possible revenues and profits. Reaching global markets means millions, hundreds of millions, and sometimes billions of individual customers or end-users, and thus information privacy and security becomes paramount. Information technology systems are often the wrapper of sensitive customer and end-user data, so the security of the information technology wrapper becomes the only barrier between your data and an army of greedy global hackers. Of course, if your information technology fabric can be penetrated, then purveyors of ransomware can simply encrypt your information technology system and its data and extract a payoff from you to unlock it. Therefore, since today's product and service innovators are directly targeting the greatest possible impact (i.e., billions of global users), it behooves innovators to design their products and services with ironclad security in mind. A traditional security engineering framework may consist of trained and certified security leads, security lifecycles, security practices, secure technologies, and a battery of security verification and validation tests (in addition to security monitoring and incident response). The cost alone of security engineering for a large information system with thousands of business requirements is economically prohibitive. However, it is a good idea for innovators to understand the values, principles, frameworks, methods, practices, tools, technologies, measurements, and overall implications of traditional security thinking. This is a loaded statement, since few enterprises execute a comprehensive battery of traditional security thinking principles and practices. So, this is a good time to start. The good news is that lean thinking involves greater simplicity, smaller batches, and a fewer number of lean business experiments. The bad news is that lean business experiments will be exposed directly to the enterprise's market-facing production fabric in seconds, minutes, hours, and days. So, the security of the innovatively new products and services must be considered early and often. This means lean canvases, UX maps, and business experiments must take security very seriously early and often. The good news is that modern cloud and DevSecOps technologies enable the ability to create extremely secure lean business experiments, oftentimes in the form of tightly controlled microservices.

22. Sim·ple • Plans (*sĭm'pəl • plănz*) Scheme, strategy, blueprint, schedule, timeline; To establish, apply, and utilize extremely simple, nearterm, and uncomplicated plans as possible for creating ecosystems of innovatively new products and services

- ✓ Smallest possible scope.
- ✓ Fastest possible deliveries.
- ✓ Business experiment oriented.

A very important principle of SAFe DX is to devise very simple plans. Much is at stake in today's global marketplace, and the risks to success have never been greater. A traditional thinker may attempt to address this risk by creating a mountain of business requirements, a 5-10-or-15-year enterprise architecture, and a tightly packed integrated master schedule (IMS) to ensure full utilization down to the second with split precision earned value management (EVM). However, over-scoping, immense complexity, frozen infinite production queues, and full-utilization are an instant death knell, not only in today's Internet of Things (IoT)-based marketplace, but they've always been ineffective even in last century's glacial marketplace. The antithesis or antidote to traditional thinking is lean-agile thinking, which is based on small batches, limited-WIP, and market-driven business experiments to "validate" assumptions rather than having big six consulting firms divine mountains of business requirements 5-10-or-15-years in advance. It's just so simple to synthesize endless assumptions in the form of bottomless business requirements specifications when your big six business analyst graduated at the top of his or her Ivy league class. However, as the old saying goes, "Any intelligent fool can make things bigger and more complex but takes a touch of genius and a lot of courage to move in the opposite direction!" Hence, we now have the principle and practice of simple plans (i.e., a simple, sparsely populated lean-agile roadmap with a few small business experiments for the next few months or quarters). Simple three phase now-next-later roadmaps are popular among the lean-agile community with the now phase representing the next 30 to 90 days, followed by the next and later phases. Only the first phase is committed to, while the second two phases are only speculative based on the results of the first phase. Each phase has to be sparsely populated to allow time for a few rinse-and-repeat cycles because a single lean business experiment is just that (i.e., experimental), and it is not a single one-and-done, fire-and-forget hypothesis test. For instance, let's say a small lean-agile team can solution one well thought out business experiment in five business days. That's a reasonable assumption. Well, it may take two or three more weeks or design sprints to refine the experiments based on the statistical hypothesis tests (and the design sprints may not be contiguous). Therefore, it's completely unnecessary to plan 12 consecutive unique business experiments in a single 90-day period, when the lean agile team may only be able to complete 3 or 4 business requirements during that time period. Worse yet, don't make the mistake of planning 5 to 10 feature sized business experiments in a single two-week Scrum sprint—That's simply insane! And definitely don't attempt to use lean-agile frameworks such as SAFe as a feel-good substitute for integrated master schedules (IMSs) and plan 700 business experiments in 90 days—That's simply lunacy!

23. A·dap·tive • Plan·ning (*ə-dăp'tĭv • plă'nĭng*) Pliable, elastic, flexible, adjustable, malleable; To create frequent, nearterm, and evolving plans for creating ecosystems of innovatively new products and services based on everchanging market needs

- ✓ Evolutionary thinking.
- ✓ Frequent replanning cycles.
- ✓ Allow room for growth/emergence.

A closely interrelated principle of SAFe DX is the practice of adaptive planning. A well thought out plan should not exceed more than a single business quarter or 90-day period. And, of course, it should be sparsely populated, aptly known as lean thinking, to avoid the common error of full utilization in the name of squeezing blood out of a turnip (i.e., squeezing every penny out of your high-tech sweat shop). However, it is acceptable to forecast a "few more" lean business experiments beyond the first planning period, which commonly known as a lean-agile product roadmap. However, a roadmap and integrated master schedule (IMS) with earned value management (EVM) should NOT be conflated (i.e., confused as one in the same which is a common mistake of traditional thinkers and leaders). Yes, it's true that both the IMS and EVM practices emerged from the public sector under rather auspicious circumstances (i.e., they never truly worked from their inception but were promulgated under false pretenses anyway). However, most Western public sector organizations have renounced legislating these stone age practices, so if your firm is still practicing the art of IMSs and EVM, you're simply a sadomasochist, because few public sector buyers, at least the smart ones, mandate these practices anymore. So, first, check your sources "before" claiming the devil made you do it! Conversely, a lean-agile roadmap (i.e., not an IMS) may consist of multiple time periods such as months or quarters with some level of forecasting. However, only the first time period has any degree of fidelity, while subsequent time periods are highly speculative. And each time period itself should be considered flexible, malleable, adaptable, modifiable, and changeable. A lean-agile team should not hesitate to pivot after a single business experiment should it prove to be ineffective or a better market need emerges. Oftentimes, the best alternative market need emerges as a result of pursuing the primary hypothesis. For instance, let's say you're Tesla, and your R&D division is testing a business hypothesis for a slightly newer nickel-cobalt-aluminum oxide chemistry battery with a 20% lower manufacturing cost and a 10% longer life expectancy. However, on day two of its design sprint, innovators identify six alternatives (i.e., hydrogen fuel cells, lithium-sulfur, graphene supercapacitors, redox flow, aluminum-graphite, and bioelectrochemical batteries), but the chief engineer rejects the alternatives in favor of the status quo (which is a common outcome). Chances are, one of the six alternatives offers a superior value proposition, so Tesla is now faced with several alternatives (i.e., stick with the status quo, select one of the alternatives, put one of the alternatives on a future plan, or nix the alternatives altogether because they are not currently on the roadmap). If Tesla was using adaptive planning, they may consider one of the alternatives now or in the near future, but the most likely scenario is that the design sprint members will leave Tesla to exploit the rejected alternatives.

24. Micro • Time • Horizons (*mī′krō • tīm • hə-rī′zənz*) Span, range, timeline, schedule, interval; To apply the shortest possible strategies, tactical plans, or timelines necessary to create a simple ecosystem of innovatively new products and services

- ✓ Shortest possible delivery cycles.
- ✓ Plan for rapid delivery/obsolescence.
- ✓ Rapid throwaway business experiments.

Another closely interrelated principle of SAFe DX is the practice of micro time horizons. That is, lean-agile plans or roadmaps should never extend beyond about two or three time periods, quarters, or even months. This is especially true in the case of the high technology industry, where innovative technologies emerge every few weeks or months. By the time a firm charters a project to create an innovatively new product or service and selects a technology or set of technologies upon which to build or assemble that product, chances are the technology components or stack are already 5-10-or-15 years behind the power curve (i.e., they are obsolete). Now, let’s say the firm codifies these obsolete technologies into a 5-10-or-15-year integrated master schedule (IMS). Although most high-tech projects have less than a 10% chance of completing on-time, on-budget, or ever, let’s assume this one does complete on-time. However, when it does, its technology stack is now 30 years old. Let’s put this in practical terms. Let’s say you are going to create a taxi company and it’s the year 2020. However, you choose automobiles manufactured in 2005 and buy 400 today. However, it takes you 15 years to establish your firm and turn your first profit, so it’s now the year 2035. So, your automobiles are now 30 years old. Seriously, who is gonna want to request the services of a 2005 automobile in 2035? With the rate of technology change that’s simply ridiculous! Therefore, innovators must take these factors into consideration, plan micro time horizons, try out a few business experiments, aim for maximum market impact now, rinse-and-repeat, and pivot when necessary. Good, bad, or indifferent, don’t commit the next 15 years of your schedule to 15-year-old technology. Firms like Apple, Samsung, Google, and Amazon are well aware of these dynamics, along with every other high-tech firm. Most long lead item hardware-intensive smartphone manufacturers release entirely new models twice a year and are capable of doing so in 90-day cycles, but simply want to recoup manufacturing costs without cannibalizing sales. Software vendors are not bound by long manufacturing cycles and can release thousands of business experiments to their enterprise’s production fabric each and every day. It’s nearly impossible to keep up with the rate of innovation in the smartphone space, just ask Blackberry, Motorola, or Nokia who simply couldn’t keep up. Imagine how much more difficult it is to keep up with the rate of technological innovation in the software space with Google and Amazon making tens of thousands of tested commits to their production fabric each day. What’s the bottom line? Firms must now think in terms of micro time horizons, strip long lead items to their bare minimum features, stop creating 5-10-or15-year-or-more IMSs and ditch the EVM that public sector agencies rarely mandate anymore, focus on smaller time horizons, and stop making the classical mistake of inserting hundreds of business experiments into 90-day plans for the sake of full utilization and economic efficiency.

25. Lim·it·ed • WIP (*lĭm′ĭ-tĭd • wĭp*) Waste, excess, surplus, overflow, inventory; To minimize unnecessary, unneeded, and excess materials, work, scope, and traditional bureaucratic delivery methods for creating innovatively new products and services

- ✓ Viciously limit work-in-process (WIP).
- ✓ Use single-tasking and simple processes.
- ✓ Dramatically reduce utilization for uncertainty.

An elemental principle of SAFe DX is the practice of limiting WIP (i.e., work in process or work in progress), which is often equated to the inventory necessary to develop a new product or service. Let’s say you’re manufacturing refrigerators, then WIP may be steel or plastic to make the doors and frames, electric motors to power the coolers and freezers, plastic or metal tubing, insulation, wiring to distribute the power, an electronic computer to monitor its health and status, plastic components for shelves and drawers, etc. Let’s say, you want to make $50 million in profit from refrigerators. The per unit cost manufacturing cost may be $800 and the profit is $200. So, you’d have to sell 250,000 units, or you’d need $200 million in labor and inventory. You may want to limit your WIP, make smaller batches of 25,000 units, and only hold $20 million in labor and inventory. If the market for refrigerators plunges, then you’ve gained about $5 million in profits along the way while saving $180 million in labor and inventory. Another way to look at WIP is as waste. In lean thinking, seven forms of waste are overproduction, inventory, motion, defects, over-processing, waiting, and transport. Traditional thinkers believe that 5-10-or-15-year integrated master schedules (IMSs), enterprise architectures, business requirements, documentation, processes, governance boards, gate reviews, checklists, manuals and procedures, etc. are value adding components. Oftentimes, 80% to 90% of an innovation product or service’s cost is tied up in this WIP. The theory is that all assumptions must be recorded forever, and the value is in capturing them (just in case something was missed). It’s sort of like packing 50 pairs of socks for an overnight trip (just-in-case). This is the kind of overthinking originated in the public sector where infinite resources were available to capture excess waste. We now know that 95% of documented business requirements, assumptions, predictions, and other divinations of the future are never used at all (and that basic ratio still holds true today). Even 95% of lean business experiments fail to realize their basic expectations. So, if you're planning a billion-dollar project, and $950 million is tied up in unfounded assumptions, then that overprocessing is simply WIP or waste. Even lean-agile projects tend to over-plan a bit. They determine the average rate of velocity in story points—Let's say a story point is about a day's worth of effort just for argument's sake—Determine the number of staff hours in a 90-day period—And, then multiply that by the amount of story points. For instance, let's say you have a 125 person ART, therefore they can product 8,125 story points fully loaded, or 6,500 story points at 80% loading. Beautiful, then they further speculate they can complete 130,000 story points in 20 quarters or 5 years. Oh boy, now we just need to compute the cost per story point and do earned value management (EVM) to ensure our high-tech sweat shop feature factory is not goofing off! Seriously though, full (story point) utilization is the worst kind of WIP.

26. Ex·cess • Ca·pac·i·ty (*ĕk'sĕs' • kə-păs'ĭ-tē*) Time, room, buffer, space, margin; To build in excess time, resources, and availability for creating innovatively new products and services by limiting their scale, scope, size, complexity, and frequency

- ✓ Smallest possible workloads.
- ✓ Allow time for unplanned requests.
- ✓ Allow buffer for risks and uncertainty.

A critically important principle of SAFe DX is the practice of building in excess capacity. That is, leaving a little room for creativity, innovation, uncertainty, risk, and gradually uncovering hidden, tacit, and inexpressible market needs. This is a very controversial principle, because, quite frankly, few people even know what it means to build in excess capacity, unless you've been through the fire or valley of death when creating innovatively new products or services. SAFe recommends that lean-agile teams be loaded to 80% utilization or 20% excess capacity, which is simply nonsense. No self-respecting computer engineer, data center operator, or system administrator would load a computer to 80% utilization. In fact, that is the maximum load that a network or data center operator would load any one device (i.e., microprocessor, RAM, storage, network switch, gateway, high-performance cluster, etc.). Beyond about 80%, the data center or any one of its components would immediately become saturated and the entire data center would be subject to immediate failure (as a data center is only as strong as its weakest link). With cloud computing, a cloud engineer would simply program a trigger to automatically provision an additional virtual component if any one of the critical data center components reached 80% utilization. Now, let's apply this to creating innovatively new products and services. You certainly don't want to create a 90-day plan with an 80% load. Think about a data center or cloud designer creating a server farm with an 80% load and no failover. The data center would instantly be crushed. Modern IT data center managers understand that IT workers should be loaded to less than 50% due to the frequency of unplanned outages, network vulnerabilities, and other component failures. With cloud systems, these failures may be minimized and managed, but not eliminated if the cloud system is not properly provisioned with the correct thresholds, fail over contingencies, and elastic provisioning options. Take a lesson from IT data center managers and reconsider loading your innovators to 80%. Once again, like a data center, any one component, i.e., lean-agile team that reaches or exceeds an 80% loading threshold will cause the entire team of teams or agile release train (ART) to come to a crashing halt. Remember the tried-and-true principles of lean thinking (i.e., keep-it-simple, keep-it-small, make-it-easy, small batches, limited WIP, sparsely populated lean-agile roadmaps, room for adaptation, emergence, and growth). Uncertainty is uncertainty—That is, it's uncertain—Innovators simply don't know what they don't know. A lean-agile plan is a more like a small business experiment or hypothesis test unto itself. Innovating is not like making hamburgers, you simply cannot divide the amount of available time by the length of time it takes to make a single hamburger. Innovating is not like a fast-food restaurant or feature factory. Start treating your team of teams or ART like an innovation team and less like a fast-food hamburger joint loaded to 80% utilization.

27. In·no·va·tion • Time (*ĭn'ə-vā'shən • tīm*) Term, space, period, stretch, interval; To build in excess time, resources, and availability for exploring options, alternatives, changes, and innovations to existing products, services, and service products

- ✓ Built-in time for innovation.
- ✓ Built-in time for experimentation.
- ✓ Built-in time for creative exploration.

Another closely related principle of SAFe DX is the practice of planning innovation time. That is, allocate a generous proportion of each day, week, month, and quarter specifically for innovation. Why not load each day to 50% for planned business experiments and leave the other 50% of the day for innovators, who may use the time for training, learning, prototyping, ideating, and other growth areas. Part of the innovation time can be used for exploring design alternatives to the main business experiment, or other alternative paths just-in-case the main hypothesis leads to a dead-end. Don't tax an individual's day beyond about 3 or 4 pm or tax weekends, because that is when people innovate. Fridays are a good day for all-day innovation time. Procrastinators do their personal business Monday through Thursday as their personal innovation time and save Fridays for all-day SAFe ceremonies (and sometimes Saturday and Sunday too). This is simply nonsense. Do your work Monday through Thursday, allocate Fridays for an all-day innovation buffer, and do not load an 8-hour day with Skype ceremonies. Skype is a doubled-edged sword as it is the ultimate lean-agile technology enabler for flexible, just-in-time innovations. The only problem with that is that people will schedule non-stop Skype meetings, come in late, leave early, or have them late in the day after they've taken their dog to the vet. Yes, a dog is more important than a SAFe meeting or innovator's time! Worse yet, people will schedule a Skype meeting and suddenly cancel it 10 minutes after it starts because the dog needs a bath. Some companies turn off their email servers on Fridays to prevent people from interrupting employees during their innovation time. Some people just can't sit still, it takes a lot of trust, discipline, and energy to use your innovation time wisely. Select your employees wisely, because, quite frankly, if people can't keep their scheduled Skype meetings, you probably can't trust them with innovation time. SAFe has a built-in innovation and planning sprint as the last iteration in a 90-day program increment. This is a bit of a misnomer because most organizations could not accept an entire sprint devoted to innovation and planning for the first decade of SAFe's existence. Now that SAFe is becoming the norm, innovation and planning sprints are also becoming the norm, well, sort of. The innovation and planning sprint is now hell sprint where product managers devise a five-year plan, give it to teams 10 minutes before program increment planning, firms schedule non-stop employee development meetings, and programs even do major system upgrades, system deployments, and other heavy training during this period. The hell, we mean innovation and planning sprint turns into 10 straight 12-hour days. RTEs re-swizzle teams, reset velocity clocks, and plunge teams into forming, storming, performing, and norming hell. Scrummasters get assigned multiple teams and have to create densely packed plans for two or three teams without additional breakout time.

28. Sim·ple • Vi·su·al·i·za·tions (*sĭm'pəl • vĭzh'o͞o-ə-lĭ-zā'shənz*) Visible, pictorial, graphical, illustrative, iconographic; To communicate with images, pictures, and graphics to plan, design, and create innovatively new products and services

- ✓ Visual information radiators.
- ✓ Simple visual short-term plans.
- ✓ Simple user experience mapping.

Another key principle of SAFe DX is the practice of applying simple visualizations to plan, design, and implement innovatively new products and services. We don't mean 5-10-or-15-year integrated master schedules (IMSs), enterprise architectures, or business requirements notations. These may be single page lean canvases, product roadmaps, user experience maps, or Kanbans. It's easy to blow up a Kanban to the size of a small IMS or shrink the size of the work items to cram dozens of items on a single page (i.e., Kanban doesn't cure stupidity). Size and complexity represent WIP, waste, assumptions, full utilization, smoke-and-mirrors, vaporware, illusion of productivity, irreconcilable dependencies, and opacity. The closer one gets to full utilization, the longer wait, cycle, and lead times increase (and even approach infinity in the form of frozen queues). This isn't to say that the initial state of lean UX visualizations may not get complex during the brainstorming process, but that the raw ideas must be condensed and simplified into simple visualizations representing the current or best plan, set of hypotheses, or WIP. Remember, a plan is just WIP until it is realized as a finished solution. Traditional thinkers believe that complexity in the form of massive WIP represents value (i.e., a foot high stack of business requirements is valuable). But until a single business requirement is delivered, it is simply waste and has no value. It's sort of like a backlog of orders—They have no value until they are delivered, billed, and paid for. Sometimes buyers default and don't pay for their orders, so even deliveries may not count as value until the account receivables have been paid. However, don't confuse WIP with account receivables, because the latter implies a delivery has been made, where the former does not (therefore, the former has no value at all). Many lean-agile tools are undergirded by scalable databases and represent lists of bottomless queues with infinite WIP, complexity, and wait times. These are anything but simple visualizations. A simple visualization is just that, simple, sparsely populated, and has an overall feeling of satisfaction. If you are an innovator and have 70 things on your queue for the week, that is anything but a satisfying feeling and may even paralyze you with fear, anxiety, and lack of productivity, further elongating wait and cycle times. But, if an innovator has a handful of tasks per day or week, then the innovator may be more psychologically predisposed to finish early in order to free up the rest of the week for personal innovation and discovery time. Yes, there are the procrastinators that will take advantage of a simple work queue to conduct personal affairs early in the week and complete their work items on a Friday or even Sunday night. It's not unusual to see email requests for work on Sunday at 9:00 pm when the procrastinators try to complete their week's work before Monday morning's staff meeting. Watch out for epic-sized features masquerading as weeklong design sprints. Burning people out does just that, burns them out. Just don't do it!

29. Vir·tu·al • In·ter·ac·tions (*vûr'cho͞o-əl • ĭn'tər-ăk'shənz*) Online, network, distributed, electronic, computer; To communicate using the Internet, intranet, or teleconferencing system to collaboratively create innovatively new products and services

- ✓ Use of distributed teams.
- ✓ Use of distributed technologies.
- ✓ Use of global distributed talent and people.

An emerging principle of SAFe DX is the practice of virtual interactions. Early in the lean-agile movement of the 1990s, face-to-face interactions were treated as gospel. You simply were not being lean nor agile if the team wasn't in the same room (i.e., no walls, doors, cubes, offices, hotel suites, etc.). People could simply converse with one another to communicate their needs, quickly collaborate and brainstorm simple designs, and even code and test them in a few hours or days. Rich, high-context verbal face-to-face communications were superior to 5-10-or-15-year waterfall project management, systems engineering, and software development process and document intensive lifecycles. Facial expressions and body language could also be perceived and interpreted in real time as valuable communication cues. If the team was small and collocated, then even ancillary conversations could be overheard and acted upon (no utterance or body language expression goes to waste in a small-collocated team). However, as scientists have already determined, small collocated teams naturally organize themselves into hierarchies, pecking orders, cliques, and clans, and senior personnel or extroverts may actually suppress or cancel out the inputs of the rest of the small lean-agile team. What's the use of having a small five-to-seven-person team, if the alpha male or female does all of the thinking for the rest of the group and suppresses their inputs. This is a common real phenomenon that must be managed properly. In the 1960s and 1970s these were called Chief Programmer Teams, where the chief did all of the thinking, and the subordinates did the bidding of the chief, right, wrong, or indifferent. However, in the post-modern lean-agile world order, we now realize that the greatest value lies in the inputs of ALL team members and innovators, not just those at the top of the pecking order (i.e., chief programmers). We've tried to mitigate the intimidation factor or suppression of positive and negative inputs with lean-agile methods like Scrum and roles such as Scrummaster, who exist in part to help ensure equal decision-making rights among the team members. However, this only goes so far. In the end, the alpha male or female becomes the Scrummaster or suppresses the facilitation efforts of the Scrummaster to silence the inputs of the rest of the team. While virtual teams and interactions were once viewed as vastly inferior to rich, high-context face-to-face communications, they are now viewed as a means to minimize the suppression by alpha males and females. It doesn't eliminate intimidation, but it does minimize its effects. Furthermore, virtual technologies such as Skype, lean-agile tools, virtual whiteboards, and other asynchronous technologies enable greater minimization of the old fashioned pecking order that once threatened to undermine innovation especially in the Far East. Even Skype in a face-to-face setting offers some advantages to near virtualization (i.e., minimization of intimidation by alphas and maximization of Scrummaster facilitation abilities).

30. A·non·y·mous • In·ter·ac·tions (*ə-nŏn'ə-məs • ĭn'tər-ăk'shənz*) Unnamed, unknown, incognito, nondescript, unidentified; To maximize the impact of creating innovatively new products and services by focusing on ideas and outcomes vs. people

- ✓ Focus on outcomes instead of people.
- ✓ Use of virtual brainstorming tools/technology.
- ✓ Minimize interference from individual personalities.

A closely interrelated principle of SAFe DX is the practice of anonymous interactions. In traditional systems and early lean-agile methods, the focus was on live face-to-face interactions, communications, and body language, which were optimized for these types of innovation events. They came in the form of quarterly planning, sprint planning, retrospectives, design sprints, pair programming, mob programming, and other lean UX events. Scrummasters, servant leaders like SAFe RTEs, and other facilitators were trained, certified, and culled to help small, medium, and large-scale teams collaborate together with one big voice. Some enterprises can easily organize a single meeting with 100 to 200 people and make everyone in the room feel like they are heard. Sometimes, it's not necessarily to gather inputs, but to promulgate a final solution to which a community feels ownership (i.e., I was there when the declaration of independence was signed, therefore its my constitution, even if my inputs nor voice were never even considered). Apart from stone age facilitation principles and practices like these, newer technologies are emerging to allow anonymous interactions so that everyone has a voice. Once such example is a virtual Internet whiteboarding tool called Mural. People can visit a single virtual whiteboard from anywhere in the world anonymously, regardless of race, religion, creed, gender, age, and other demographics, and directly contribute to an idea. A facilitator may ask anonymous innovators to post ideas on a virtual whiteboard, vote on them and sequence them, and down select them to the smallest set of business experiments to be conducted in the form of a lean-agile roadmap. If the ideas come from everyone all over the world, then the maximum possible innovation range is considered. With anonymous interactions, the highest-paid-person's-opinion (HIPPO) effect is minimized, managed, and/or eliminated. For decades, decision makers have been trying to eliminate cognitive bias of enterprise executives that blind them to rational decision-making processes and outcomes and undermine their very success. Well, anonymous interactions may do just that, because the lowest-paid-person-in-the-room also has a voice alongside the HIPPO. This applies to lean-agile events and ceremonies such as value stream mapping, lean canvases, solution and product management, lean UX, sprint planning, and more importantly retrospectives, which may be the single most valuable tool in the lean-agile arsenal. Early, fast, and continuous improvement is the foundation of lean-agile thinking, it is often relegated to the last second, it is hurried, and only HIPPOs typically have a voice. With anonymous interactions, everyone on an innovation team can provide many inputs to retrospectives in just a few seconds, whether they are good, bad, or indifferent. Modern Internet tools such as Poll Anywhere even parse the inputs and rank them based on frequency, so group consensus can be achieved rapidly without a biased post-retrospective analysis.

31. Fast • Meas·ur·a·ble • Feed·back (*făst • mĕzh'ər-ə-bəl • fēd'băk'*) Data, facts, figures, statistics, information; To collect measurable end-user, customer, and market feedback on innovatively new products and services as quickly as possible

- ✓ Gather fast customer/market feedback.
- ✓ Gather qualitative/quantitative information.
- ✓ Improve products-services based on feedback.

A quintessential principle of SAFe DX is the collection of fast measurable feedback. The value proposition of traditional thinking rests upon volumes of WIP in the form of integrated master schedules (IMSs), business requirements, and multi-decade long, multi-billion-dollar enterprise architectures. Traditional thinking is the antithesis of fast measurable feedback. First generation lean-agile methods tried to break the curse of traditional thinking by shipping working innovations to markets, customers, and end-users at the greatest possible velocity, thus becoming high-tech sweatshops or feature factories. Maximum velocity is the mantra of lean-agile feature factories. However, modern lean-agile thinking involves moving away from one-and-done, fire-and-forget feature factories (i.e., shipping crappy code), but rather a series of small business experiments, with the expressed purpose of not shipping crappy code, but gathering valuable business intelligence from live customer and end-users. So, in modern lean-agile thinking, the value proposition is in the market, customer, and end-user feedback, not in the payload (i.e., small business experiment). It's sort of like sending up a weather balloon, deep space probe, or other scientific measurement instrument (i.e., the value is not in the measurement package, but the data and information the instruments collect). Lean-agile thinkers, unlike traditional thinkers, realize they cannot divine the future for decades by hiring big six consultants to manufacture a mountain of IMSs, business requirements, and enterprise architectures. In other words, modern lean-agile thinkers comprehend the nature of business intelligence, it most often exists in the form of tacit, hidden, intangible, and inexpressible market, customer, and end-user needs, utterances, and unspoken body language that can only be elicited or transformed into explicit, tangible form by collecting measurable feedback. Yes, speed or velocity is still important to modern lean-agile thinkers, but not through fully utilized feature factories and sweat shops, because they only lead to large batches of untestable, crappy code. Speed is not obtained through full or maximum utilization, which comes from more than a century, if not millennia of traditional thinking, because the users of first generation lean-agile methods were traditional thinkers, but from smaller batch size, lower WIP, and excess capacity (i.e., small sparsely populated business experiments). When small lean-agile teams swarm, collaborate, or mob program on small business experiments, as in the case of Google's one-week design sprints, then magic happens, high-impact multi-billion-dollar product and service innovations emerge that reach billions of global end-users at a fraction of the price and cost of traditional multi-billion projects. Furthermore, the feedback doesn't have to be raw sales or profits, although that's nice, but are leading indicators (i.e., likes, downloads, registrations, referrals, and total user volume that can be monetized through indirect sales).

32. Im·pact • O·ri·ent·ed • Meas·ures (*ĭm'păkt'* • *ôr'ē-ənt'ĭd* • *mĕzh'ərz*) Metrics, models, results, measures, outcomes; To collect data and information reflecting positive or negative outcomes of innovatively new products and services vs. process data

- ✓ Focus on leading vs. lagging indicators.
- ✓ Focus on outcome vs. output measures.
- ✓ Focus on results vs. vanity measurements.

A critically important principle of SAFe DX is use of impact-oriented measures, which are known as outcome vs. output measures. The hallmark and foundation of traditional thinking is output measures (i.e., number of processes, documents, pages, lines-of-code, epics, features, user stories, velocity, effort, defects, compliance, integrated master schedules, work packages, audits, gate reviews, tools, technologies, WIP, etc.). First generation lean-agile methods shifted the focus to outcome such as number of user stories and story points completed, which is not the same as delivered nor measured. In other words, first generation lean-agile methods simply accelerated the process of creating inordinately large batches of crappy code for a small handful of testers to evaluate. Early practices such as continuous integration tried to undo this debacle by having programmers compile, build, smoke test, and automate as much unit testing as possible. However, this was often only hit-and-miss, piecemeal, and done on developer environments, a process which completely disintegrated in the production environment. Furthermore, the larger the code base, the more difficult it became to continuously integrate in under a lifetime. Again, the mathematics of large batches reared its ugly head in the form of frozen queues. Programmers finally got it right with continuous delivery and DevOps, by automating the process of delivery to an enterprise's production fabric with a lot of automated production tests. For instance, Google programmers make over 60,000 code commits per day to version control, run hundreds of millions of production tests on those commits, and make tens of thousands of code deliveries to their production fabric. The key of course is the modularization of the code deliveries into fully independent microservices (i.e., small batches). Basically, each code commit may be a complete batch unto itself with only a few thousand tests. However, Google code commits are not just one-and-done, fire-and-forget crappy code snippets, but small business experiments, with the expressed purpose of collecting market, customer, and end-user feedback (from millions of simultaneous global end-users). Therefore, the purpose of these lean experiments is not full utilization of Google developers or the realization of high-tech sweatshop feature factories, but to evaluate the impact of the experiments on leading indicators such as Pirate metrics. These may include awareness (impressions, click through rate, attention-minutes, site visits, likes, social shares, social impressions, podcast impressions, etc.), acquisition (leads, subscribers, downloads, chats, captured emails, etc.), activation (signups, loss-leader product sales, freemium customers, replies, etc.), revenue (customer acquisition cost, trial to paid conversion, first purchase, etc.), retention (customer lifetime value, net promoter score, churn, modified churn, expansion revenue, etc.), and referral metrics (net promoter score, referrals, social shares, awareness metrics, etc.).

33. Con·tin·u·ous·ly • Im·prov·ing (*kən-tĭn'yo͞o-əs-lē* • *ĭm-pro͞ov'ĭng*) Tune, temper, perfect, enhance, strengthen; To frequently change, modify, and simplify the process of creating innovatively new products and services along with the results themselves

- ✓ Frequent process-product-service improvements.
- ✓ Avoid static long-term processes-products-services.
- ✓ Delegate process improvements to lowest possible level.

The foundational principle of SAFe DX is the practice of continuously improving. That is, to continuously improve all SAFe DX principles, as well as the outcomes of small business experiments and their results to create the most innovatively new products and services possible. The foremost principle is to wean leadership off of traditional thinking and first-generation lean, agile, and traditional hybrids, i.e., teach true lean-agile leaders to stop mixing and matching integrated master schedules (IMSs), earned value management, volumes of business requirements, and multi-decade long multi-billion-dollar enterprise architectures together with lean-agile frameworks. In other words, stop using lean-agile methods and frameworks to form high-tech sweatshops and feature factories to reach full utilization, reduce the cost of story points to its lowest possible price threshold, and deliver thousands of stories forecasted years in advance with split-second EVM timing. This is simply not lean-agile thinking, industry leadership, nor the purpose of lean-agile frameworks, although this has become the norm. Instead, continuously improve and reduce batch sizes, WIP, utilization, dependence upon traditional thinking, and density of lean-agile product roadmaps, 90-plans, and two-week sprint plans. Learn to do more with less, speed up by slowing down, and improve quality with simplicity. These are the hallmarks of lean-agile leaders, enterprises, and innovation projects. True lean-agile leaders shift the focus away from fierce Western individualism. That is, using divide-and-conquer approaches, parsing out user stories to individuals, and applying wicked multi-tasking and full-utilization to execute densely populated IMSs, product roadmaps, 90-day plans, sprint plans, and Kanbans. Instead, reduce batch size, WIP, and utilization, plan a few business experiments, and solution them "together" in small teams that can pull small, sparsely populated batches through Kanbans, Scrumbans, Scrumboards, 90-day SAFe plans, and lean-agile roadmaps pretty quickly. Reducing complexity speeds up throughput, while increasing complexity slows and even stops throughput. People are rather clever and game velocity metrics by boasting how fast they can throw WIP over the wall into its next unfinished state. Remember, a batch in an unfinished state is simply a queue of WIP, and filling queues to capacity is the death knell of innovation, whereas full queues were the hall mark of traditional thinking where nothing is delivered to the enterprise's production fabric, nothing is measured, and markets are simply lost (forever). Continuously improving means to find the goldilocks zone, sweet spot, or center of percussion when it comes to balancing batch size, throughput, fast feedback, rinsing-and-repeating, and pivoting to a new business experiment. The goal is not to deliver an ineffective feature fast but find one that delivers the best performance. This means sifting through many business experiments quickly without being wed to any single product or service solution (not even for a microsecond).

SAFe DX Principles Summary

So, what exactly have we learned in this short treatise on how to apply SAFe DX principles and practices to create innovatively new products, services, and service products? Well, first and foremost, we've learned that lean-agile leadership is the linchpin upon which SAFe DX principles and practice rest. That is, without strong lean-agile leaders, high-impact benefits to enterprises, markets, customers, end-users, and bottom-line profits cannot and will not be achieved. We've known for more than half a century that it is the top-level executive (i.e., CEO) who matters the most, so if your CEO is not a lean-agile leader, then SAFe DX is just another unrealized pipe dream. The reason the CEO matters, is because they surround themselves with like minded people, hire corporate executives and directors to further their agendas—Hopefully, SAFe DX agendas—And they create reward systems and incentives to ensure their strategic plans are successful—Hopefully, SAFe DX strategies. Although the fate of lean-agile leadership, and SAFe DX for that matter, lies directly in the hands of the top-level executive, enterprises must hire, appoint, and grow lean-agile leaders at all levels of the enterprise, including middle managers, front-line managers and supervisors, and technical leads and the innovators themselves. Many corporate change initiatives quickly die-on-the-vine in the hands of bureaucratic middle managers from the stone ages who are impervious to executive level direction, modernization of their infrastructures, cries of dissatisfied markets, customers, and end-users, and relentless sudden erosion of corporate revenues, profits, and overall value. Part of this involves keeping middle management bureaucracy a little bit lean, but certainly not non-existent. Yes, we live in the age of radical corporate downsizing where management ranks have been stripped to their bare minimum and program managers must essentially manage hundreds of people without a support staff clearly violating span of control, breeding tyrannical behaviors, and forming immediate immunity to change due to intense cognitive blindness. However, lean-agile leadership doesn't stop there, but involves training, growing, certifying, and promulgating lean-agile thinking and SAFe DX principles practices at the lowest layer of the enterprise fabric itself. That is, our employees are our 21st century knowledge workers, they are the engines and innovators of our lean-agile teams, and if our innovators are impervious to lean-agile thinking then we are done, we've failed, and we simply cannot move the innovation needle forward. It's simply not enough to put a Scrummaster in charge of traditional thinkers and expect our lean-agile innovation teams to succeed. Lean-agile thinking must saturate the entire fabric of our enterprises, organizations, non-profits, public sector agencies, corporations, and small businesses.

33 Principles and Practices for Improving Scaled Agile Framework (SAFe) Developer Experience

1. **Lean-Agile Thinking**—To apply small batches of WIP-limited business experiments to tease out tacit, inexpressible needs
2. **Lean-Agile Leadership**—To exemplify, promulgate, and reward principles of lean thinking at all levels of an enterprise
3. **Lean-Agile Frameworks**—To apply proven highly-cohesive lean-agile principles, practices, and reference models
4. **Decentralized Decisions**—To empower teams with the authority to select day-to-day options, choices, directions, etc.
5. **Empowerment Oriented**—To grant equal decision-making rights, powers, and authority to select and improve innovations
6. **Innovation Oriented**—To establish an operating model, culture, and system to create innovatively new products/services
7. **Keep It Simple**—To establish a basic, easy-to-use system for creating innovatively new products and services
8. **Make It Painless**—To establish a lean-agile operating model that is not overly burdensome, complicated, nor bureaucratic
9. **Make It Fun**—To establish a culture, lean-agile system, and innovations that are engaging, pleasant, and enjoyable
10. **Small Batch Size**—To create innovative products/services that are small, simple, uncomplicated, quick, and easy to deploy
11. **Small Experiments**—To create innovative products/services as small hypothesis tests designed to gather rapid feedback
12. **Small Deployments**—To develop innovative products/services that can be rapidly and frequently delivered early and often
13. **Small Feature Teams**—To apply small teams of people with complementary technical skills, abilities, and talents
14. **Small Project Size**—To apply the smallest number of people necessary to create a small, highly cohesive ecosystem
15. **Small Product Size**—To create the smallest possible ecosystem of innovatively new products and services
16. **Lean Product Management**—To apply a formal process for scoping extremely small batches innovative products/services
17. **Lean User Experience**—To apply user experience design techniques for creating innovative products and services
18. **Lean Design Sprints**—To quickly scope innovative products and services customers need and want in small timeboxes
19. **Design for Scale**—To design and develop innovatively new products/services for the largest number of simultaneous users
20. **Design for Test**—To design and develop innovative products/services to be tested early, continuously, and automatically
21. **Design for Security**—To build in information, system, and application security when creating innovative products/services
22. **Simple Plans**—To establish simple, near-term plans for creating ecosystems of innovatively new products and services
23. **Adaptive Planning**—To create frequent, near-term, and evolving plans for creating innovatively new products and services
24. **Micro Time Horizons**—To apply the short strategies, tactical plans, or timelines to create innovative products/services
25. **Limited WIP**—To minimize excess materials, work, scope, and traditional WIP when creating innovative products/services
26. **Excess Capacity**—To build in excess time, resources, and availability for creating innovatively new products and services
27. **Innovation Time**—To build in excess time for exploring innovations to existing products, services, and service products
28. **Simple Visualizations**—To communicate with images to plan, design, and create innovatively new products and services
29. **Virtual Interactions**—To communicate using Internet to collaboratively create innovatively new products and services
30. **Anonymous Interactions**—To maximize the impact of innovative products/services by focusing on ideas and outcomes
31. **Fast Measurable Feedback**—To collect measurable feedback innovative products and services as quickly as possible
32. **Impact Oriented Measures**—To apply leading indicators to measure impact of innovative products and services
33. **Continuously Improving**—To frequently simplify the process of creating innovatively new products and services

Beyond the nearly impossible task of creating an enterprise-wide psychology, consciousness, and culture of lean-agile and SAFe DX thinking, enterprises must apply lean-agile thinking principles, practices, methods, frameworks, tools, technologies, and metrics. That is, many organizations and countless individuals have spent millions and sometimes billions of dollars formulating well-crafted lean-agile thinking frameworks, and some have even outfitted these with principles of business agility, lean portfolio management, lean-agile architecture, systems engineering, and UX practices, business experimentation, and even DevSecOps.

So, it becomes more important than ever to take a second fresh look at lean-agile frameworks such as SAFe, DaD, S@S, Enterprise Scrum, LeSS, RAGe, SAGE, Spotify, Scrum, Scrumban, XP, DevSecOps, Lean Startup, Startup Way, Design Sprints, Lean UX, business experiments, and similar proven frameworks and methodologies. Although organizational change takes time, many frameworks such as these—Especially SAFe—not only contain many best practices and recipes for instant success, but they contain training wheels for novices and beginners to apply with immediate beneficial outcomes. Organizational change comes last in SAFe, not first, as SAFe's first goal is to enable any enterprise, be it traditional or ad hoc, to realize the immediate benefits of lean-agile thinking, practices, and principles. However, even frameworks like SAFe can be easily abused by well-meaning people to realize traditional principles and practices such as integrated master schedules (IMSs), enterprise architectures, and volumes of business requirements documents written by big six consultants decades ago. In other words, they'll use SAFe to build an IMS, plan for full utilization of all personnel resources, and then hold Scrum of Scrums meetings to check on earned value management (EVM) measures of hundreds of people every day. In doing so, they simply achieve what we've always done since the dawn of the industrial age, create fully utilized high-tech sweatshops and feature factories, reduce the cost of labor and story points to their lowest possible values, plan every minute of an employee's workday with a stopwatch like Frederick Taylor, Watts Humphrey, and Jack Welch and allow productivity and innovation to sink to the lowest possible levels while our global competitors run by us like Japan did in the 1970s. Therefore, true lean-agile leaders must stop designing high-tech sweat shop feature factories, cramming thousands of story points into our backlogs, and squeezing blood out of a turnip with EVM and daily Scrum of Scrum meetings. Instead, true lean-agile leaders do the exact non-intuitive opposite, dump 20th century traditional thinking principles and practices which never worked in the first place and embrace the higher ideals of lean-agile thinking.

So, what exactly are these so-called values, principles, and practices of lean-agile thinking? Well, apart from selecting and applying proven lean-agile thinking frameworks like SAFe—Correctly that is—These involve applying principles and practices of business agility, lean portfolio management, lean-agile product management and UX, and emergent and evolutionary architectures. Furthermore, this includes principles and practices such as business experiments, DevSecOps, and even flexible, adaptable, and low-cost lean-agile technologies such as cloud computing, which is the ultimate medium to realize the outcomes of lean-agile thinking. More importantly, lean-agile leadership means going back to lean-agile basics, fundamentals, and rudimentary blocking and tackling. It means to apply extremely small (micro) batches vs. infinite complexity, limiting WIP vs. integrated master schedules (IMSs) and earned value management (EVM) to achieve full utilization, and it means deploying small infrequent business experiments directly to an enterprise's production fabric to collect immediate market, customer, and end-user feedback (vs., delivering user stories in a one-and-done, fire-and-forget fashion). A user story is not an explicit systems requirement, but a simple agreement to have a conversation, form a business experiment to test a hypothesis, and uncover the true, hidden, tacit, and inexpressible market, customer, and end-user need in a rinse-and-repeat fashion. Therefore, the so-called values, principles, and practices of lean-agile thinking involve everything but what they've become in the early 21st century. They are not a means to form high-tech sweatshop feature factories to burn down 5-10-and-15-year IMSs at maximum velocity and full-utilization with split second EVM timing. Conversely, the values, principles, and practices of lean-agile thinking involve forming a few business hypotheses, forming a sparsely populated strategy in the form of a lean canvas and product roadmap, creating sparsely populated 90-day lean agile plans, and empowering small front-line lean agile teams to swarm on possible solutions in short time-boxed intervals. Timeboxing these events may sound a little antithetical to the values and principles of lean-agile thinking, but mechanisms like one-week design sprints are a nice forcing function, set of guidelines and guardrails, and anti-lock braking system to prevent innovation teams from over-scoping, gold-plating, and gold-fleecing their business experiments. Furthermore, design sprints force innovation teams to deploy their business experiments directly to the enterprise's production fabric for immediate validation by markets, customers, and end-users, thereby serving as a basis for a few well-timed rinse-and-repeat cycles to get it just right and find the goldilocks zone, sweet spot, or center of percussion with respect to end-user needs.

Perhaps SAFe DX is the next step in the evolution of lean-agile thinking, which first emerged in the 1940s and 1950s, giving rise to the Toyota Production System (TPS). However, as few people realize, Western information technology consultants converted TPS principles into agile methods such as Scrum, Kanban, Scrumban, Extreme Programming (XP), Continuous Integration (CI), Continuous Delivery (CD), DevSecOps, Feature Driven Development (FDD), Dynamic Systems Development Methodology (DSDM), Crystal Methods, Lean UX, and many others. From there, this gave rise to agile project management and scaled agile frameworks such as SAFe, DaD, S@S, Enterprise Scrum, LeSS, RAGe, SAGE, Spotify, and finally Business Agility, Lean Startup, Startup Way, Design Sprints, and Business Experiments. The bottom line is that TPS is now obsolete, while many Western consultants are still promulgating its outdated principles and practices, which not even the Japanese do anymore. Western lean-agile frameworks are for knowledge workers who create global innovations in minutes, not factory workers making new cars every four or five years. However, at the heart of TPS were ideas like flexible manufacturing, bottoms up innovation, short-term adaptable production cycles, continuous improvement, respect for people, small batches, limited WIP, and something far less than full utilization. Western firms introduced a bit of time-driven profit motivation and innovation into the laissez faire Japanese culture and TPS equation that is wickedly immune to change. Lean-agile thinking does not mean memorizing and applying hundreds of statistical manufacturing equations as lightweight lean-agile frameworks like Scrum, SAFe, and Design Sprints have proven. This isn't to say that a few statistical hypothesis tests aren't needed to measure the final results of business experiments, but perhaps Western consultants can simplify these to help fast moving teams quickly measure the impacts of their innovations without a grad degree in statistics. Let's stop synthesizing years of untestable business requirements, cramming these into integrated master schedules (IMSs), high-jacking lean-agile frameworks like SAFe to realize them at maximum velocity, and filling everyone's days with scrum of scrums meetings and story point counting to take our temperature as often as possible. Let's take a giant step backwards, slow down, breathe deeply, ponder the meaning of lean-agile thinking, reduce batch sizes, limit WIP, lower utilization, reward teamwork, foster emotional intelligence vs. constant backbiting, and enable our enterprises, portfolios, solutions, projects, and teams to successfully create innovatively new products and services just one step ahead of expanding global competition. Our front-line innovators are our greatest gifts, so let's start treating them that way with SAFe DX principles and practices.

Further Reading

- Baker, J. (2017). *Developer experience (DX): Devs are people too*. Retrieved August 20, 2021, from http://bit.ly/3sBfv4k
- Belkind, L. (2020). *Building developer-first products with developer experience (DX) 101*. Retrieved August 20, 2021, from http://bit.ly/3AS8ZJw
- Cavalcante, A. (2019). *What is DX*? Retrieved August 20, 2021, from http://bit.ly/3ghxPdZ
- Chang, A. M. (2019). *Lean impact: How to innovate for radically greater social good*. Hoboken, NJ: John Wiley & Sons.
- Coyler, C. (2020). *What is developer experience (DX)*. Retrieved August 20, 2021, from http://bit.ly/2WaEQ9a
- DuVander, A. (2021). *Why developer experience matters*. Retrieved August 20, 2021, from http://bit.ly/3Bfc0nx
- Fagerholm, F., & Munch, J. (2013). *Developer experience: Concept and definition*. Retrieved August 20, 2021, from http://bit.ly/3B6z5st
- Gajda, M. (2021). *Why should we care about developer experience (DX)*? Retrieved August 20, 2021, from http://bit.ly/2W95SOe
- Jarman, S. (2017). *The best practices for a great developer experience (DX)*. Retrieved August 20, 2021, from http://bit.ly/2UELpk0
- Kaye, B. & Jordan-Evans (2014). *Love em or lose em: Getting good people to stay*. San Francisco, CA: Berrett-Koehler.
- Knapp, J. (2016). *Sprint: Solve big problems and test new ideas in just five days*. New York, NY: Simon & Schuster.
- Luca, M., & Bazerman, M. H. (2020). *The power of experiments: Decision making in a data-driven world*. Cambridge, MA: MIT Press.
- Memarovic, N. (2020). *Developer experience (DX) 101: How to evaluate it via diary study*. Retrieved August 20, 2021, from http://bit.ly/3j4uEIg
- Messinger, J. (2020). *Why your API needs a dedicated developer experience team*. Retrieved August 20, 2021, from http://bit.ly/380zjVt
- 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 Group.
- Radonich-Camp, D. (2020). *The importance of developer experience for your digital product's success*. Retrieved August 20, 2021, from http://bit.ly/3D3U5BL
- Reselman, B., & Broberg, M. (2020). *Developer experience: An essential aspect of enterprise architecture*. Retrieved August 20, 2021, from http://red.ht/3sDmA4v
- Rico, D. F. (2016). *The 10 attributes of successful teams, teamwork, and projects*. Retrieved September 26, 2016 from http://davidfrico.com/teamwork-attributes-2.pdf
- Rico, D. F. (2016). *The 12 attributes of successful collaboration between highly creative people*. Retrieved February 29, 2016, from http://davidfrico.com/collaboration-attributes.pdf
- Rico, D. F. (2018). *Lean & agile contracts: 21 principles of collaborative contracts and relationships*. Retrieved June 29, 2018, from http://davidfrico.com/collaborative-contract-principles.pdf
- Rico, D. F. (2018). *Using SAFe 4.5 to transform a $200 million U.S. healthcare portfolio*. Retrieved November 19, 2018 from http://davidfrico.com/safe-case-study-ii.pdf
- Rico, D. F. (2019). *32 attributes of successful continuous integration, continuous delivery, and DevOps*. Retrieved September 27, 2019, from http://davidfrico.com/devops-principles.pdf
- Rico, D. F. (2019). *Business value of lean leadership: 22 Attributes of successful business executives, directors, and managers*. Retrieved April 27, 2019, from http://davidfrico.com/rico19a.pdf or http://davidfrico.com/lean-leadership-principles.pdf
- Rico, D. F. (2019). *Evolutionary architecture: 24 principles of emergent, organic, and highly adaptive design*. Retrieved September 3, 2019, from http://davidfrico.com/evolutionary-architecture-principles.pdf
- Rico, D. F. (2019). *Piloting the scaled agile framework (SAFe) in a top 10 U.S. energy firm*. Retrieved May 15, 2019, from http://davidfrico.com/x-safe-case-study-iii.pdf
- Rico, D. F. (2020). *32 principles and practices of highly successful SAFe implementations*. Retrieved February 16, 2020, from http://davidfrico.com/safe-principles.pdf
- Rico, D. F. (2020). *Business value of lean thinking: Capitalizing upon lean thinking principles to rapidly create innovative products and services*. Retrieved January 29, 2020, from http://davidfrico.com/rico20b.pdf or http://youtu.be/wkMfaPAxO6E
- Rico, D. F. (2020). *Using large solution SAFe in a high-profile U.S. civilian public sector agency*. Retrieved January 29, 2020, from http://davidfrico.com/y-safe-case-study-iv.pdf
- Rico, D. F. (2021). *32 principles and practices for a highly successful agile contract statement of work (SOW)*. Retrieved March 22, 2021, from http://davidfrico.com/agile-sow-principles.pdf
- Rico, D. F. (2021). *32 principles and practices for maximizing the ROI of SAFe program increment (PI) planning*. Retrieved March 14, 2021, from http://davidfrico.com/safe-roi-principles.pdf
- Rico, D. F. (2021). *32 principles and practices for successful distributed lean-agile programs, projects, and teams*. Retrieved May 7, 2021, from http://davidfrico.com/agile-distributed-principles.pdf
- Rico, D. F. (2021). *32 principles and practices to successfully transition to U.S. DoD cloud computing data centers*. Retrieved June 12, 2021, from http://davidfrico.com/dod-cloud-principles.pdf
- Rico, D. F. (2021). *Business value of organizational agility*. Retrieved March 26, 2021 from http://davidfrico.com/rico21a.pdf or http://davidfrico.com/value-of-business-agility.pdf or http://youtu.be/HOzDM5krtes
- Rico, D. F., Sayani, H. H., & Sone, S. (2009). *The business value of agile software methods: Maximizing ROI with right-sized processes and documentation*. Ft. Lauderdale, FL: J. Ross Publishing.
- 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.
- Schrage, M. (2014). *The innovator's hypothesis: How cheap experiments are worth more than good ideas*. Boston, MA: MIT Press.
- Thomke, S. H. (2020). *Experimentation works: The surprising power of business experiments*. Boston, MA: Harvard Business Review.
- Ward, C. (2020). *An introduction to developer experience (DevEx, DX)*. Retrieved August 20, 2021, from http://bit.ly/389b6vX
- Zeman, J. (2021). *Good developer experience*. Retrieved August 20, 2021, from http://bit.ly/3B0PVsB

Case Studies of Successfully Using SAFe DX Principles and Practices to Create Innovative Products and Services

- Global Energy Firm. *This is a case study of a global energy firm that selected the Scaled Agile Framework (SAFe) as part of its digital transformation initiative to restart its information technology (IT) department after decades of malady, entropy, and neglect. This, of course, was a global energy monopoly, their customers were captive, and their competitors were few. It quickly became one of the largest global energy monopolies after a series of corporate level mergers and acquisitions in rapid succession. With each merger and acquisition, corporate level IT budgets and departments were simply merged and retired in order to save what was perceived as an unnecessary IT expense. The energy business is a capital-intensive business largely realized through controlling existing nuclear power plants, hydroelectric power plants and dams, industrial power plants, coal fired power stations, solar and wind farms, and, of course power grids and distribution channels such as substation control centers, transmission lines, and other local power lines. Of course, there are other energy industry functions like trading energy futures, and of course selling energy to customers such as states, industrial plants, and individual consumers. Much of today's energy grid pre-exists in the form of capital assets, and most corporate investment is devoted to capturing and controlling these assets, vs. creating new forms of energy, improving the infrastructure itself, or modernizing its IT infrastructure to help manage the assets belonging to any single corporate monopoly. In fact, modernizing one's IT infrastructure is very low on the list of corporate priorities, and investments in IT modernization are very lean, thin, and sometimes non-existent. However, in this case, after a successful series of mergers and acquisitions, this energy firm decided to restart its stalled IT infrastructure with a digital transformation initiative. It's first move was to hire a visionary IT leader who was up-to-speed on modern IT trends. As such, he labored to improve its consumer billing systems with a digital transformation initiative and retained the services of one of the largest big six consulting firms. The firm suggested modernizing its customer billing systems with mobile apps, creating a lean UX department, and applying lean-agile thinking principles, practices, and frameworks. The consulting firm helped its energy client right-size its digital transformation initiative for its paper-thin IT budget. Value stream mapping was performed along with user experience mapping, and more than 20 teams were chartered to address strategic and tactical elements of the value stream that would provide the greatest impact for the least amount of investment. Sometimes, these pain points required the services of more than one lean-agile team to form small portfolios or ecosystems of highly cohesive products and services. After some consternation, three small lean-agile teams were formed into a small development value stream, they were taught SAFe, and they were empowered to form sparsely populated lightweight 90-day lean-agile roadmaps and plans. Small batches of business experiments enabled them to quickly create innovative web services for their firm.*
- Public Sector Firm. *This is a case study of a top 15 U.S. public sector supplier in their industry sector. It became the fastest growing firm in its class by successfully capturing large, multi-billion-dollar public sector IT contracts. Although many public sector agencies were no longer in the business of legislating information technology (IT) delivery frameworks, traditional thinking was etched deep in the culture of this firm as it was for most of its competitors. However, most public sector suppliers began adopting lean-agile methods about a decade after most commercial firms, and then adopted larger-scale lean-agile frameworks like SAFe around mid-2010, catching up with many commercial firms. Fortunately for this firm, it cut its teeth on Scrum on a small number of teams building up some measure of competency, before making the transition to SAFe. However, it wasn't an easy transition to SAFe as they increased the number of teams applying Scrum by five times in the transition to SAFe, including all non-software teams. In other words, most teams were learning Scrum for the first time just as they were learning SAFe making the transition a bit rough. To add insult to injury, their technology stack was obsolete and unstable, increasing the number of unplanned outages which were constantly disrupting their SAFe 90-day plans, because they were planning to full capacity and utilization. Growing system instability motivated them to load teams to 80% capacity, which was a bit unnerving to their government client who was used to having suppliers plan and operate at full-capacity and utilization for over 70 years. However, it was too little too late, as SAFe's recommended 80% loading of lean-agile teams was similar to full capacity in an unstable system, and among innovation teams where uncertainty is extremely high. Many teams lost the appetite for Scrum and SAFe and convinced leadership to allow them to operate without SAFe plans or ceremonies at all. Unplanned outages were directly traceable to the fact that this firm was only applying the bare minimum number of SAFe practices necessary such as program increment (PI) planning, Scrum ceremonies, and frequent Scrum of Scrums to track the growing number of system failures. In other words, most teams were skipping Scrum retrospectives and were not conducting Inspect and Adapt (I&A) ceremonies to perform a root cause analysis and uncover the causes of the frequent system outages. Without a Solution or Product Management Team, Lean UX principles and practices, and value stream mapping exercises, they were simply blind to the fact that their technology stack was now 20-30 years old and very brittle. After three years of applying SAFe, they began applying more SAFe practices, hiring more lean-agile leaders, reducing capacity to proper innovation levels, and using small, specialized agile release trains (ARTs) of innovation teams to pioneer DevSecOps pipelines. In other words, by employing SAFe DX principles and practices of smaller batches, limited WIP, low utilization, business experiments, and teamwork, they rapidly created an innovatively new architectural runway to stem the tide of the rapidly aging IT infrastructure.*
- Cloud Services Provider. *This is a case study of an information technology (IT) services supplier, who successfully transitioned to lean-agile frameworks and methods like Scrum and SAFe just-in-time to convert an end-of-life traditional brick-n-mortar data center to a cloud-based system. This Top 5 IT services supplier in its class was steeped in traditional thinking but was one of the earliest adopters of lean-agile frameworks and methods like Scrum and SAFe. This project fell by the wayside and missed its corporation's lean-agile initiatives. Realizing that its IT data center contract was ending, it reached back to its corporate parent and took the initiative to ask for help transitioning to lean-agile frameworks. Lean-agile consultants were sent in for early discovery sessions and determined this IT project was ripe for a SAFe adoption. External SAFe coaches were hired to help with its transition to SAFe practices. The hope was that they could win the next phase of the contract, which was to transition the aging brick-n-mortar IT stack to cloud-based technologies by showing the initiative apply modern IT delivery frameworks like SAFe. The SAFe coaches followed the SAFe implementation roadmap, trained all of the IT data center operators in SAFe and Scrum, and kicked off its first SAFe planning event rather quickly. Although its leaders expressed interest in transitioning to SAFe, they just didn't apply the necessary upfront effort for a properly planned SAFe launch. So, the SAFe coaches were pressed to train 7 or 8 Scrum teams of about 80 people at the last minute constituting its Agile Release Train (ART). Although a lean-agile product management team didn't exist, the SAFe coaches rapidly groomed a program backlog of features for its first SAFe planning event which is very common. The SAFe coaches suggested delaying the SAFe launch due to lack of preparedness, but the clock was ticking, and its leaders insisted upon a quick SAFe launch. Program increment (PI) planning was conducted, the new Release Train Engineer (RTE) improvised as quickly as he could in real-time, and the SAFe coaches backed him up by coaching the teams through SAFe PI planning and preparing them for Scrum launch. With the SAFe launch behind them, the RTE relegated the SAFe coaches to monitoring daily standup meetings while making unilateral SAFe process decisions in a vacuum for the first 90 days. Although the ART got off to a slow start, the SAFe coaches convinced the RTE to include them in continuously improving the ART's performance. The ART delivered 90% of its SAFe PI objectives by the third PI. Although the supplier did not win the next phase of the contract, its personnel were hired by the new cloud services provider, which chose to continue using SAFe to rollout the new cloud computing infrastructure with the old RTE leading them. Throughout the SAFe rollout, the SAFe coaches successfully coached the RTE to apply SAFe DX principles and practices vs. traditional thinking, establishing a solid architectural runway for its cloud computing phase. In other words, the SAFe rollout was so successful, the client asked the follow-on consulting firm to continue applying SAFe and keep the prior staff along with its RTE.*