AGILE THROUGH & THROUGH:

LEARNING TREE EXPERTS TALK
REQUIREMENTS | DESIGN | TRANSFORMATION

An eBook from Learning Tree International

LEARNING TREE™ INTERNATIONAL

1-800-843-8733 • LEARNINGTREE.COM

AGILE THROUGH & THROUGH:

LEARNING TREE EXPERTS TALK
REQUIREMENTS | DESIGN | TRANSFORMATION

TABLE OF CONTENTS

ABOUT THE AUTHORS

ALAN O’CALLAGHAN

Alan is an experienced Lean-Agile transformation consultant, Scrum coach, and software architect who works with large corporations and start-ups alike, helping them to gain fluency in their Agile approach. As a Certified Scrum Trainer with Scrum Alliance he is qualified to train and certify Agile leaders in organizations to establish Agile practices and raise the performance level of their Agile teams and the business value of their Agile projects, drawing on his own extensive experience in the Product Owner and Scrum Master roles.

Alan is principal product owner at Emerald Hill Limited as well as the instructor and course author for some of Learning Tree’s most popular Agile training courses, including the Scrum Master Certification training course.

TIMOTHY GUAY

Tim Guay is a leader in Lean Agile best practices with a proven track record as a team member, trainer and coach. He is experienced in introducing Agile and Lean into organizations, providing Agile and Lean training and coaching, as well as both Agile Engineering and DevOps best practices. Tim is well-versed in migration from traditional waterfall to Agile software development processes at both the team and enterprise level.

He has delivered Agile training and coaching to a wide-range of organizations from start-ups to Fortune 50 corporations since 2002.

MAURICE HAGAR

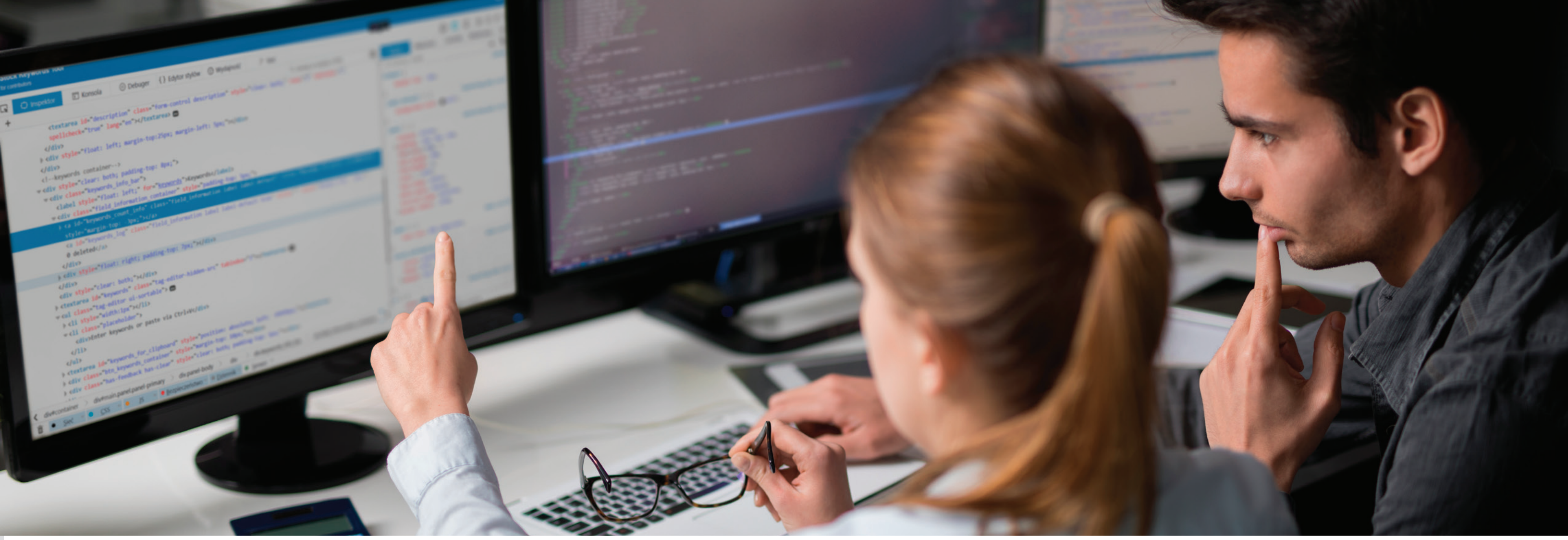
Maurice has been coaching and training Agile transformations for Fortune 100 companies, government agencies, and the U.S. military since 2005. He has trained hundreds of teams and led multiple enterprise-wide Agile transformations. Mr. Hagar has been a course instructor at Learning Tree for more than 12 years.

In order to bring you well-rounded Agile insights from a variety of perspectives, the authors featured in this eBook are experts in a variety of job roles. Alan O’Callaghan is a Business Analyst, Maurice Hagar is a Project Manager, and Tim Guay is a Software Designer.

INTRODUCTION

Agile is a mindset, applicable to all types of organizations, that enables continued customer value-focused product delivery in a world of ever-evolving requirements through the collaborative effort of self-organizing, cross-functional teams. From developers to business stakeholders, everyone in the organization must work together with an Agile mindset to continually realign the product/service with customer needs and company goals.

In this eBook, some of Learning Tree's leading Agile experts and instructors share their thoughts on several topics related to adopting and "being" Agile in an enterprise setting — from requirements, to design, to scalability and governance.



HOW DO YOU CONQUER JUST-IN-TIME REQUIREMENTS?

Alan O’Callaghan

Detailing all requirements up front can be a waste of effort in the development of software products. The average churn after a document has been signed off is said to be about 35%, although many software professionals report a much higher figure.

By the time the requirement is reached, there’s a good chance it will have either significantly changed or disappeared completely from the to-do list. The effort spent in eliciting needs, analyzing them, and documenting the system requirements that meet those needs has been wasted.

That, in turn, entails a cost: the salaries of the professionals involved for a start. And now at least part of that effort and cost must be spent again in understanding the new situation.

THREE RESPONSES

There are broadly three responses to this problem:

- **Do nothing. Just take the hit. In today’s commercial environment that’s not sustainable. The cost is too high.**
- **“Freeze the requirements.” It is more typical than you might think.** I’ve heard that order given a good few times in my career. And every time it has been about as successful as King Canute’s instruction to the sea to stop the tide coming in. Why? Because of change — change in technology and, more importantly, change in business conditions — is our only constant. Freezing requirements might make a delivery manager’s life easier, but it always means less business value will be produced.
- **The best response is one that utilizes just-in-time requirements engineering: detailing out only the system requirements that are on the short-term build horizon.** Longer-term requirements-driven “planning” isn’t planning at all really. It produces gross speculations dressed up as plans. There are too many unknowns at the beginning to really plan in detail.

See next page to continue article.

DRIVING JUST-IN-TIME REQUIREMENTS WITH USER STORIES

The Scrum Guide says that, "Higher ordered product backlog items are usually clearer and more detailed than lower ordered ones." This implies just-in-time requirements, but the Scrum framework doesn't describe any process or mechanism for achieving this. This is entirely in line with the philosophy of Scrum. It is for self-organizing teams to figure out what is best for their own context.

Undeniably, the popular choice is User Stories. These originated from XP and, in particular, from Ward Cunningham and Kent Beck. Maybe nine in every ten Agile teams use stories as Product Backlog Items (PBIs). Used properly, they are very effective in driving just-in-time requirements. But in my experience, they are rarely used properly.

STORIES ARE NOT SYSTEM REQUIREMENTS

One of the biggest mistakes teams make is to confuse User Stories with System Requirements documentation. This either leads to stories that themselves are long documents, or to there being no documentation at all other than the first versions of the stories — often just three-line statements of what a user wants.

PBIs are not requirements themselves; they represent requirements. They are items in a list ordered by the Product Owner. So, when stories are used as PBIs they are not requirements either. Actually, it is more accurate to say that they are not system requirements.

They are short statements of user needs (the IIBA calls these stakeholder requirements) for which the development organization is charged with providing a solution. This requires intense collaboration and multiple conversations between customers and developers to decide which solutions fit those needs best.

Stories are designed to trigger those conversations, while system requirements documentation should be their outcome. How much documentation and what form it should take is again a decision for the development team itself.

TIMELY CONVERSATIONS

Once it is properly understood that stories are there to trigger conversations, the team can decide how and when is the best time to have them. Of course, when a story is about to be pulled into a Sprint for development, it must be "ready." In other words, the development team must understand it well enough to start work on it straight away. **Most high-performing Scrum teams will have enough "ready" stories at the top of the Backlog to occupy them for between 1 and 3 Sprints.** This basically means that all the necessary conversations will already have taken place.

The remaining stories in the Backlog are often called epics. This just means they are too big and vague to be considered ready for development. There are more conversations to be had.

VISION

The Product Owner decides which epics to decompose first for more detailed conversations. Ultimately, it is the Product Owner's responsibility to get stories or PBIs to "ready." Their judgments about the relative importance of the items are reflected in her ordering of the Product Backlog. These are themselves made in the context of the understanding of the Product Vision. **One of the most difficult aspects of the Product Owner role is to keep the big picture of the vision in the minds of the development team even while they are focused on the short-term Sprint Goal.** That's a skill that must be learned — but it's a lot easier than creating the up-front requirements document.

NEXT STEPS

If you'd like to know more about just-in-time requirements, these courses/products can help you:

- **Effective User Stories** • *Course 4598*
- **User Stories: Getting to Ready** • *Course 3653*
- **Certified Scrum Product Owner** • *Course 1814*

LEARNING TREE INTERNATIONAL

IS AGILE DESIGN AN OXYMORON?

Timothy Guay

There is a persistent myth that Agile Design is an oxymoron, as there is no place for design in Agile; that our design and architecture will magically emerge as we code. This opinion is a natural reaction against the waterfall Big Design Up Front (BDUF) mentality, but it is a false and even dangerous opinion.

DANGEROUS IN THAT THE AD HOC "DESIGN" THAT COMES FROM DOING NO DESIGN RESULTS IN CODE THAT IS:

- Hard To Reuse
- Hard To Understand
- Hard To Maintain
- Hard To Integrate
- Riddled With Technical Debt

Coding without doing any design work is not Agile; it is hacking pure and simple and you will end up with an instant legacy system.

See next page to continue article.

SO HOW DO WE APPROACH AGILE DESIGN?

Firstly, we maintain our focus on business value as the key driver for our design. A Fit-For-Purpose, business-value-driven design results from an ongoing collaboration between the team, the customers, and other relevant stakeholders.

Secondly, the design is owned collectively by the team, and the code is co-owned by the designer via their design. The developers will own certain design aspects at the unit and user story level. The designers and architects serve as design SMEs to the developers, as well as facilitate collaborative design workshops.

TRUE TO THE AGILE PRINCIPLE THAT TECHNICAL EXCELLENCE ENHANCES AGILITY, AGILE DESIGN EMPHASIZES EXCELLENCE IN DESIGN. THIS RESULTS IN:

- **A robust, scalable, and modifiable architecture and design**
- **A design that results in testable, maintainable, and reusable code**
- **A design that results in code and tests that are amiable to Continuous Integration (CI) and Continuous Deployment (CD)**

It is important to avoid the dangers of over-designing and over-documenting, so it is key to do only just-enough design that will ensure a coherent Fit-For-Purpose design. Documentation should be low-fidelity, so put Visio away as it is far easier to collaborate around a whiteboard than a computer screen. It is also quicker to sketch on a virtual whiteboard than to create a diagram in Visio, then just snap a picture and attach it to the user story.

AGILE DESIGN CONCEPTS AND TECHNIQUES

The goal is to do just enough design and architecture up front, with detailed design being done as needed during backlog grooming. Key design concepts and techniques are summarized in the table below.

CONCEPT / TECHNIQUE	DESCRIPTION
SHEARING LAYERS	**Each architectural layer changes/evolves at different speeds.** Keep them loosely coupled as tight coupling results in rigid, hard to modify and hard to understand design.
MODEL STORMING	**Collaborative brainstorming** designed to quickly develop high-level architectures and designs using techniques such as CRC Cards or Domain-Driven Design.
INTENTIONAL ARCHITECTURE	**Leverage common architectural patterns, constraints, and implementation technologies to optimize usability, extensibility, performance and maintenance.** Addresses both business and technical requirements. Prove out by creating a Minimum Viable Architecture.
ARCHITECTURAL RUNWAY	**Provides sufficient architecture** to support the incorporation of near-term product backlog items without needing architectural refactorings.
MINIMUM VIABLE ARCHITECTURE	**The minimum architectural implementation required to prove out an end-to-end architectural design.**
DESIGN PATTERNS	**Provide proven, reusable design building blocks.** Developed by looking at the common characteristics of solutions to related problems.
DESIGN PRINCIPLES	**Describe Object Oriented design best practices.**
DESIGN FOR TESTING	**Design principles focusing on designing code that is optimized for automated testing.**
ACCEPTANCE TEST-DRIVEN DEVELOPMENT	**Use to do behavior-focused design at the user story level. The Given-When-Then structure:** • **Guides** the design of the application flow and state changes • **Helps** identify the inputs, processing, and outputs • **Refines** the low-level design prior to coding

If you are interested in a more in-depth look at Agile Design, check out Learning Tree's Agile software design course, which also qualifies participants for the ICAgile Certified Professional in Agile Software Design certification.

NEXT STEPS

If you'd like to know more about incorporating Agile in the early stages of software design, these courses/products can help you:

- **Agile Software Design Professional** • *Course 944*
- **Agile Test Automation** • *Course 1820*
- **Impact Mapping: Focusing on Business Value in Agile Development** • *Course 3654*

JUST-IN-TIME SCALING AGILE

Alan O’Callaghan

Scaling agile is a hot topic. Courses on SAFe (Scaled Agile Framework], DAD (Disciplined Agile Development - from PMI) and LeSS (Large Scale Scrum) are becoming more popular. As large organizations move into the Agile space they want to know how to run big projects which, traditionally at least, have required multiple skillsets scattered across multiple working groups or even departments. Hence the attraction of these, and other, frameworks.

However, I have a sneaking suspicion that many scaling efforts are misguided, and often generate more waste than value.

A RESTAURANT ANALOGY

Let me begin by introducing a restaurant analogy. Recently for the birthday celebration of an extended family member, my wife and I, together with a mob of relatives and friends, ate at a chain eatery in Coventry, England near where we live. It is one of those places where you pay a single price and then eat as much as you like from an enormous buffet. Indian, Chinese and Italian specialties are included. There is also a grill, a roast meats station, and a dessert counter. Each station has its own specialist crew of kitchen staff. They make sure that there is a constant supply of food items to keep the counters fully stocked.

In order to do this, they employ many cooks in the kitchen.

Now compare this to a much more up-market Indian restaurant that, as a couple, we more regularly favour with our custom. It has a small kitchen and a correspondingly small kitchen staff.

In either place we might eat a starter, a main and a dessert, but in our Indian restaurant the chef is cooking to-order and can start on the mains even while we are eating our starter. The other place has to have all the potential elements of the meal on offer at the same time.

INCREMENTAL VALUE DELIVERY

The point is that whether or not organizational scaling is necessary is a judgement call that depends on the timeliness of required features. The big eatery has no option but to scale its kitchen teams because all the “features” are required at the same time. Most master planned development projects have a similar constraint. ‘Big bang’ delivery means shipping all the required features simultaneously. Low value items and big value items — they all have to be delivered together. This is usually much more than a single development team of ten or fewer people (the size recommended by The Scrum Guide) can handle.

Multiple teams are required. Scaling is necessary.

We need to be careful that this old muscle memory of our pre-Agile history does not go unchallenged. Agile in general, and Scrum, uses the motto, “deliver early, deliver often”. And we don’t just deliver any old thing. We deliver the highest business-valued features first. Our customers can munch on those even while we are developing the next set of features.

Think about the difference. If the customers can only get everything at once, then there is likely to be high pressure for the earliest date for that “big bang” delivery. Until they get delivery they can extract no value. Every day they wait adds cost. From the development side, the features that take the longest to develop hold back the delivery of everything in the product. Scaling is possibly the only option.

On the other hand, if the customers are getting hold of slices of functionality incrementally, and are extracting value from using those early delivered features then they can afford to wait for the lesser-valued ones. Now perhaps the same team can deliver all the product. At the very least we can say that scaling (the use of multiple teams) is now optional.

SCALE THE MINDSET BEFORE THE FOOTPRINT

What I'm suggesting is that deciding how many teams will be needed in advance of development is not a great idea. **Big projects should, in any case, be started with just one development team.** The early technical decisions tend to be architectural ones. They will tend to impact on and constrain later choices. It is better for the architectural integrity of the product that they be taken by a single, cohesive team even as they are developing the highest valued features. It is not uncommon for a so-called 'Beach-head Team' (composed of the best programmer-architects available) to take responsibility in this way for the first release.

Judgements about whether to bring other teams to bear on the product can then be made just-in-time: in the usual spirit of Agile. Scale the mindset first. Apply the values and principles of Scrum. Decide *empirically* whether additional teams are needed to deliver the remaining features in a timely fashion. Of course, the decision-making process will require intense collaboration between the Scrum team and all the stakeholders of the project, but that's just as it should be. The extra scaffolding of any given scaling framework should never substitute for that.

Downsizing the numbers needed to deliver a product is as much a possibility as scaling as it is traditionally thought of. **At J.P. Morgan, moving from "scrum-but" to Scrum in full required removal of a whole layer of functional departments, component teams and associated level-1 managers in order to create development teams that could deliver end-to-end functionality to their customers.**[1]

There is, of course, one important factor that must be present for a single development team to be able to deliver a large, feature-rich product on its own, without involving multiple teams: it must be truly cross-functional. All the skills necessary to develop the product must be present in the team. It takes time and effort to grow such teams. The routines and practices of the team and the individuals within it have to change as they increasingly become a self-managing group. Once achieved, this level of capability opens up the options to scale or descale.

NEXT STEPS

If you'd like to know more about scaling Agile, these courses/products can help you:

- **Scaling Agile: A Guide to Meeting the Challenge** • *Course 3655*
- **Leading SAFe with SAFe 4 Agilist Certification** • *Course 1817*
- **Large Scale Scrum: More with LeSS** *(Private Training Only)*

[1] *See the report by Craig Larman and Matt Winn at www.infoq.com/articles/large-scale-scrum-jomorgan*

DON'T LET GOVERNANCE THREATEN YOUR AGILE TRANSFORMATION

Alan O'Callaghan

Governance seems to be one of those frightening words that threatens to stop an Agile transformation effort dead in its tracks. I've been hearing it whispered, and even screamed once or twice, quite a lot recently. There's no big surprise here. **As the big corporations and government agencies get increasingly fascinated by frameworks like Scrum, they are mandating their IT departments to, "go Agile" and then, sooner or later…governance!**

TYPES OF GOVERNANCE

There is operational governance represented in the defined processes that organizations and teams are expected to follow when software is in production. There is project management governance, perhaps dictated by PRINCE2 or similar, while the product is in development. PRINCE, by the way, is an acronym standing for 'PRojects In a Controlled Environment'.

Project management governance is often a subset of a wider IT governance. According to the TOGAF version 9.1, **IT governance supposedly provides the framework and structure that links IT resources and information to enterprise goals and strategies. "IT governance institutionalized best practices for planning, acquiring, implementing and monitoring IT performance..."** Standards like COBIT, which stands for Control OBjectives for Information and related Technology, might be in place. And then, of course, there is a range of issues to do with compliance to the requirements of external regulators in all public bodies, as well as commercial institutions in sectors such as insurance and banking. So, is this a case of an irresistible force (Agile) meeting an unmovable object (governance)?

WHAT IS GOVERNANCE?

Let's go back to first principles. What is governance? Here's a definition I came across recently in the TOGAF 9.1: "The discipline of monitoring, managing and steering a business to deliver the business outcomes required." As an Agilist, I have no problem with governance described in this way. An obstinate focus on the delivery of business outcomes is the very stuff of Agile. The problem is how governance is applied. Typically, specialist governance bodies are set up in a hierarchy at each level of which procedures are mandated, documents are required for sign-off, and auditing procedures abound. The Agile principle of trusting professionals to get the job done is about as welcome as a trap door in a canoe. Fear of non-compliance drives very different behaviors from those we are trying to grow with Agile.

When these kinds of bodies and procedures are imposed on software development teams there is one guaranteed result: a delay in the delivery of value to the customer (often a protracted delay at that). Phase-gates block the development path. Waiting for sign-offs builds queues of work items. Sometimes the development 'track' is idle, like a train sitting at a signal waiting for it to turn green.

"IT governance institutionalizes best practices for planning, acquiring, implementing and monitoring IT performance..."

SOFTWARE DEVELOPMENT IS DESIGNFUL

The scenarios I have just described are in no way conducive to the achievement of business outcomes. In fact, very often, it is the governance procedures which are the major obstacle to their delivery. Why is this?

Business value is much more likely to be delivered in the Agile Model because of its fast feedback cycles. These are necessary because software development is inherently unpredictable. Its practical processes are more like what goes on in the design rooms of product development than it is like the assembly lines of mass manufacture.

"Design is not passive. It is wise for designers to harmonize with the ways of nature. But the ways of nature follow context and change."[2] Mass manufacture is predictable. The uncertainties have been ironed out and removed in the design "phase". But software development is not.

While not everything in software development is design (problem analysis should shape design, after all), its 'implementation' is creative. Even when we are crafting code we are only designing the instructions that the computer will run. Design is dominated by uncertainty. New information emerges during the process itself and is incorporated as quickly as possible. Feedback cycles surface that information, allowing the development team to converge on a solution that delivers business value to the customer.

"Design is not passive. It is wise for designers to harmonize with the ways of nature. But the ways of nature follow context and change."[2]

Traditional governance procedures follow the motto "plan the work;
work the plan". They assume predictability. There are many corporate procedures where governance for predictability is appropriate. Software development is not one of them. Governance for feedback; governance for responsiveness is what is required. **In many Agile-adopting organizations, a "two-speed" solution is evident.** Where predictability reigns, traditional governance procedures are maintained. Where feedback drives success, different approaches operate.

In Scrum, the Product Owner is responsible for achieving the business outcomes inherent in the product development. The Product Owner owns the product for the business, and is accountable for, amongst other things, its alignment with the strategic goals of the business. They are also a peer member of the Scrum team. One of the reasons you rarely see formal, outlined business cases, followed by detailed business cases and post-project audits against them in Scrum is because it is unnecessary. The Product Owner role and the Sprints' inspect-and-adapt cycle takes care of that stuff in a more 'light touch' way.

Similarly, the quality of the product is best ensured by embedding testers as developers in the Scrum team. The goal of testing then itself becomes feedback. The 'Whack-The-Mole' pattern means defects can be removed by the Development team before the increment gets to the Sprint Review, let alone Release. The ping-pong between programmers and testers that so characterizes (and delays) waterfall development is eliminated by making the Scrum team responsible for quality.

While any overall solution to the governance issue will be situational, and may yet require some external oversight, the general line of approach seems obvious to me. Business stewardship and quality assurance are, in my opinion, stronger in Scrum than they are in waterfall precisely because the specialist skills needed have been brought into the Scrum team, and the team is accountable for them. The Scrum Development team is supposedly fully cross-functional: it should contain all the skills necessary to deliver the product. If that requires specialists in governance, then include them in the team.

Let the team figure out, through conversation and collaboration with whoever it needs – external regulators included – the best way to ensure the proper delivery of business outcomes.

NEXT STEPS

If you're in the early stages of getting to know more about Agile, these courses/products can help you:

- **Agile Fundamentals: Scrum Kanban, Lean and XP** • *Course 918*
- **Business Agility Accelerator** • *Course 3647*
- **Embed our Agile Coach to help guide you through training options best for your organization**

[2] *J.O. Coplien and L.Zhao Towards a General Formal Foundation of Design: Symmetry and Broken Symmetry. Monograph*

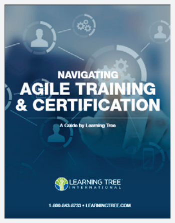
CONCLUSION

The wide world of Agile can seem like a confusing maze, but we can help you make sense of it, and work with you through the adoption of this mindset — organization-wide. Wherever you are on your journey, we can meet you there and help you run a better company, generating value at a higher rate.

Learning Tree has developed a guide to outline the most popular Agile methodologies and frameworks available today.

VIEW THE GUIDE:

LearningTree.com/AgileCertGuide



The Guide and this eBook should not take the place of a one-on-one consultation to discuss your specific goals and challenges surrounding Agile. We invite you to reach out to Learning Tree's Client Solutions Consultants so that we can better assist you in finding the Agile solution that's right for you and your team.

LEARNING TREE'S AGILE CURRICULUM FEATURES ACCREDITED CERTIFICATION TRAINING FROM THESE AGILE INDUSTRY EXPERTS:

LeanKanban University • APMG International™ AgilePM • PMI Project Management Institute R.E.P. SM • Thayer School of Engineering at Dartmouth • Scaled Agile® • ICAgile International Consortium for Agile • ScrumAlliance®

LEARN MORE AT: LEARNINGTREE.COM/AGILE
OR CALL 1-800-843-8733