A Link-Systems International, Inc. White Paper

Agile Development of Educational Software

Iterative Queuing for Web-based Learning Applications

December 28, 2012

Authored by: Robert Andersen
Director, Product Development
Link-Systems International, Inc.

Version: 1.80

4515 George Road, Suite 340
Tampa, FL 33634
813-674-0660
www.link-systems.com

Table of Contents

Table of Figures

Agile Development of Educational Software

Iterative Queuing for Web-based Learning Applications

About the Author

Link-Systems International, Inc. (LSI) is among the top ten software development companies of the Tampa Bay area. NetTutor®, LSI's online tutoring service, WorldWideWhiteboard®, LSI's online and mobile collaborative learning environment, and LSI online content services are used by educators and students around the world.

Robert Andersen is Director of Product Development at LSI, where he also functions as Queue Master for the successful LSI Agile IQ Development, based on the Scrum Development Procedure. In this paper, Andersen explains the successful application of agile programming principles and how it facilitates market readiness at LSI. He shows how Agile IQ supports LSI's longstanding collaboration with universities, community colleges, K-12 schools, and other educational programs.

In a Nutshell

The utility of a specialized agile development procedure may be seen nowhere more clearly, perhaps, than in the design, development, and implementation of effective educational software. The latter has been the business of Link-Systems International, Inc. (LSI) since 1995. Coined by Jeff Sutherland in the early 1990s, the term "agile" denotes a software development procedure that is leaner and more responsive than traditional methods. The Agile Alliance brings together the various schools of agile programming-Scrum, Dynamic Systems Development Method (DSDM), and Extreme Programming (XP), for instance. Like great educators, agile programming is flexible, realistic, and results-oriented. LSI turned to agile methods from the outset and has since evolved its own version of Scrum called Agile Iterative Queuing (Agile IQ). In brief, LSI uses test-driven programming, incorporates end-user expectations, and brings next-generation technology to educators today.

Iterative queuing is a term borrowed from software queuing applications. Agile IQ adapts Scrum development to the special needs of distance education software. This paper shows how Agile IQ embodies both agile principles and the tenets of the constructivist learning theory at the heart of all LSI products. It outlines the systematic collaboration and prioritized rounds of test-driven coding that deliver high-quality and customized software on-schedule to the academic community. Agile IQ serves LSI's vision: to bring exciting, engaging, low-overhead learning solutions to the right market at the right time.

How Agile IQ Works

The world of education is still undergoing a vast transformation, one reaching from its theory to its everyday practice. Learners everywhere have become online citizens, participants in a high-speed virtual world of communication unknown to past generations. This means that schools and courses must handle data online, promote their online presence to attract and inform potential learners, and–most significantly–present all components of the education experience online.

This does not render traditional teaching obsolete–on the contrary, web techniques give a new, round-the-clock immediacy to the learning and teaching process. However, it does mean that new software, more effective and efficient communication and learning tools, and better online learning content are constantly in demand.

For those who answer the demand for educational software development, like Link-Systems International, Inc., this means handling an unrelenting barrage of requests for systems, methods, and innovations that will make online learning, teaching, and administration more intuitive and effective. A typical development stage at LSI is really many different stages of development on a variety of products pursued simultaneously.

Each day, the development team meets to assess partial development attainments. Has the customization of the online tutoring platform for school X been completed? Is the next version of the shared online learning environment ready to release for final testing? Are tests for an important new feature of the content authoring system ready to roll out?

Each week, or every other week, depending on the task, the queue of programming tasks is frozen for a new round of development. The development of features, products, and customization is recast in a new narrative for the next round of development.

Executive management polls customer-participants for their thoughts and contributions in each area. The roll-out of completed projects begins. The Queue Master evaluates the conduct of this process and can intervene if the development team loses sight of the business aim of its work and can let management know when items added to the Queue are redundant, poorly defined, or otherwise lack adequate background, called the “story” of development.

And, through a careful review of coding accomplishments and shortcomings, as well as collaborator feedback, the Queue “iterates,” or updates. Tested components (features, product versions, or customizations) are assigned a tiered active development state, ranging from “Wait” to “Urgent.” The next development stage in each commences.

Typifying this process, every significant piece of development is time-boxed–that is, is given a specified number of development stages in which to roll out. Outside this time-box, a new narrative is fashioned to support or reject the component involved. The entire development team can focus its energies by concentrating all efforts on meeting defined programming goals and testing criteria within specific timeframes.

Above all, the result is a successful roll-out of an understandable, customer-endorsed sequence of software innovations. Each product is tested and re-tested to gauge its utility within the business model of LSI.

LSI Agile IQ software development moves through "stages," expands "story" to "epic," and delivers innovative software at every iteration of the Queue. That sounds like the logical goal of programming. It is important to understand that not all software is produced in this way; in particular, LSI is convinced that only a well-thought-out procedure that relies on programmer ingenuity and user input can adequately meet the technology needs of the academic community. The short history of software programming procedures shows how the evolution of this practice relates to other project management issues. Choosing and committing to an agile development methodology can be decisive for the ability of a software development firm to produce market-ready technologies.

Agile Software Development Procedure

History of Software Development Methodologies

Efforts to systematize the creation of software date from the advent of the computer. In the sixty-year history of software development, three major paradigms have emerged, and all of these are still in practice. Programming began as an utterly pragmatic, whatever-works procedure, passed through a period in which adherence to formal correctness and documentation was thought to be the pre-condition for effective development, and arrived at a variety of methodologies that hold the common tenet that development is realistically a shared activity of the enterprise where design is trumped in every case by product usefulness and where what is deemed useful today may be seen as redundant or otherwise less than optimal by improved technical standards in as little as eighteen months.

The first method of software development, according to which programmers just write and re-write code until the software does the required job, is called "code-and-fix." This approach may sound naïve today, but development teams also hark back to this approach as to halcyon days of freely expressed programming creativity. It certainly is still in use today. The drawback of "code-and-fix" is that if a company simply addresses programming tasks as they occur, solves them, then fixes errors in the solution, the periphery of the software development cycle will be in disorganization. The coding and error-fixing process is cut off from the critique of those who need the software. Management and designers or programmers will field complaints or requests out of step with the often brilliant accomplishments of developers. The business cycle and software development cycle are always at risk of disrupting each other.

The second paradigm of software development starts by explicitly recognizing phases of the software development life cycle. This second approach, under a number of formulations, attempts to meticulously run through each phase of development and satisfy all requirements of a project, bringing order and predictability to the process. This approach is best typified by the assignment of a name and a listing of prerequisites for the accomplishment of each step in development. Internally, no activity is complete until it has completed work on all requirements. The completion of analysis leads to design;

the design phase hands off to programmers; the code produced by programmers is run through testing. Testers use design specifications to judge whether to send software back to the programmers for corrections. Linking all activities is a chain of evidence: documentation. Designers, for instance, may use a mix of code-like summaries and visual representations such as flow-charts. Typified by the now-classical “Waterfall” procedure, development only descends to the next level of activity when the previous one is complete.

Waterfall and related methods address the intricacies of systematic, large-scale production. These second-paradigm methods, though, pose heavy internal demands and have a rigidity that can endanger the usefulness of a product. The design-program-test paradigm emerges as an end in itself, a way of satisfying requirements of development in isolation from the end-user. Unfortunately, the perceived efficiency and thoroughness of documentation can cause enough delay that by the time a usable product is released, it is out of step with a market that has moved on. In another irony, errors discovered in testing amplify development delays to the point where management may step in and impose a “code-and-fix” solution under intense time-pressure.

By the early nineties, with start-ups and Web-based software emerging and with the demonstrated difficulties of the Waterfall procedure, a new, “light-weight” philosophy of software development took root. Agile development methodology emerged from the elaboration of such a new philosophy. Practitioners such as Jeff Sutherland argued that it was pointless to separate the business goals of development from the development process. Instead, he said, software development was a collaborative activity in which design, programming, and testing were inseparably intertwined. Good design was a product of the confluence of user needs and the potential of programming rather than the other way around. In essence, development had to be ready to respond to all changes–business, hardware, or software–at a moment’s notice.

The weakness of both naïve and design-driven development was the time needed to incorporate feedback from users into code creation and testing. Agile programming earned its name by proceeding in the opposite direction: all development occurs within discrete, limited intervals of activity, incorporates testing as a starting-point, and consequently redefines project design very nearly in real time. Agile development procedures thus reconcile goals of management (on-time delivery to targeted customers) with programmers’ efforts to find the ideal implementation. The development team *as a whole* would meet its goals within the time allotted. The team would deal with unmet goals by transferring them into the next interval of activity after possibly reprioritizing them or breaking them up into smaller tasks.

This is a general description. Well-known schools of agile programming—such as XP, Scrum, and DSDM, mentioned above—have different implementations and slightly differing philosophies and methodologies. The Agile Alliance, formed in 2001, issued the Agile Manifesto to define the common features of agile methods, which is included in Appendix I. While each agile method has specific merits, the Agile IQ method of LSI is derived from Scrum.

What is Scrum?

The name Scrum is the term for the confrontations that occur during a rugby match. Players from each side tussle to take control of the game. The Scrum method is filled with such figurative terminology. Under Scrum, the description of what the software is or will do is its "backstory." Team members literally face off each day to launch a "sprint," independently assigning priorities to all tasks for the sprint of programming activity. The end of a sprint is the next milestone release or stage in completion. Unmet goals simply transfer to the "backstory backlog" and must be taken on in the next sprint. All development activity is "time-boxed," a term borrowed by LSI's Agile IQ method. This permits the time-boxing of large changes–for instance, LSI's recent and on-going HTML5 rewrite of its codebase for online collaboration tools.

Every Scrum project is realized in a number of discrete periods of maximal effort. Testing is built into the programming effort to avoid the code-and-fix cycle. Project testers instead arbitrate the acceptability of releasing what is accomplished in each unit of development effort. Testers also become expert at verifying and reshaping high-level design goals such as look and feel. The team succeeds together in a period of maximal effort or, in falling short, retells the backstory–i.e., reassigns priorities of feature development–to allow for acceptable completion at the next stage.

On every Scrum team there is a Scrum Master charged with maintaining the principles of mutual independence and collective action that typify the method. The Scrum Master also enforces the ultimate business goals by not allowing programming to dominate other components of the team. As such, the Scrum Master does not occupy or speak as a traditional representative of the process–like a manager, for instance–but as a partisan of Scrum.

By incorporating daily meetings, intense development "sprints," and a non-participant Scrum Master who reconciles interests during the sprint, Scrum effectively realizes the goals of agile programming. It also reinforces the need for management to respect the efforts of all programmers while imbuing design and coding with the goals of the enterprise.

Scrum.org is the web page of the major practitioners of Scrum. The organization's statement of principles is in Appendix II (Principles of Scrum). One long-time leader in implementing Scrum techniques, Mike Cohn, remarks on the difficulties of doing so while pointing out that "stakeholders in companies that have the transition are happy they've done so." Cohn elaborates:

> One reason…is that time-to-market is reduced when using an agile process like Scrum. This faster time-to-market is enabled by the higher productivity of agile teams, which is in turn the result of the higher quality we see in agile projects. Because employees are freed up to do high-quality work and because they see their work delivered sooner into the hands of waiting users, job satisfaction goes up. With higher job satisfaction comes more engaged employees, which leads to more productivity gains, initiating a virtuous cycle of continued improvement. (Cohn, 2010)

What is Agile IQ Development?

The history of LSI involves taking on problems that educators face in using Web-based technology. Typically, tools are available to perform tasks in general long before comparable tools are created to perform the same tasks in the much more security- and accessibility-conscious educational environment. For instance, a Learning Management System (LMS), like Blackboard®, Desire2Learn, or Canvas, performs tasks that are within the reach of a powerful spreadsheet program or database management program. The LMS functions better than the spreadsheet because it communicates safely and securely with the school's database or Student Information System (SIS) and provides storage in a format clearly dedicated to the various information-gathering tasks of teachers. Similarly, LSI produced the first proprietary synchronous virtual whiteboard for subject-based learning. That is, the LSI WorldWideWhiteboard provided the secure, teaching-enabled environment which a Web browser, for instance, could only hint at. In addition, teachers can carry on live, in-depth discussions of a specialized subject like mathematics using the actual notation proper to the subjects. Toolbars and a natural layout for text entry facilitate learning on this product.

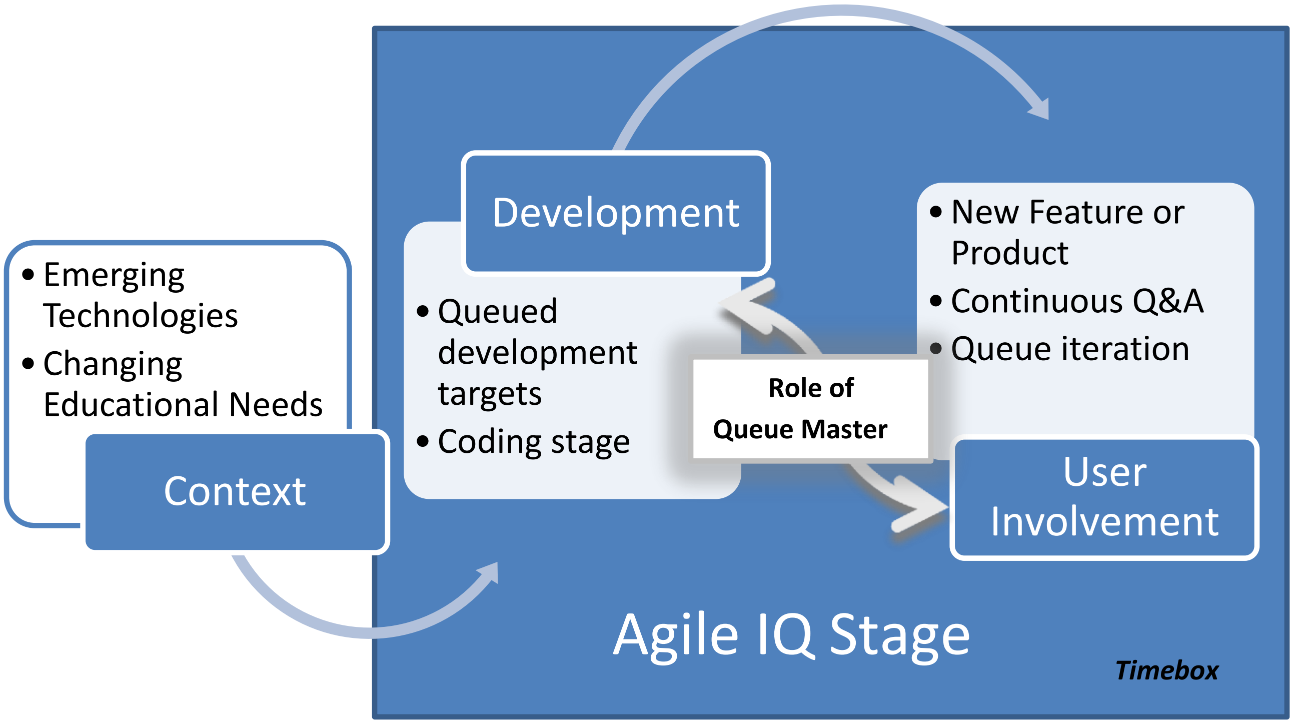

Figure 1. Agile IQ: Components of the Development Team in a Stage Timebox for Iteration of the Queue

Tens of thousands of teachers and students make use of the WorldWideWhiteboard at any given time. The LSI twenty-four hour customer service sites and the efforts of the LSI Sales Development staff to reach out to new educational institutions generate a constant stream of user input about the system. This puts sales and operations personnel in an excellent position to comment meaningfully on user experience and to help prioritize development. Under these conditions, only an agile development procedure was appropriate because agile methods cycle business goals directly into the production of software.

The centralized character of LSI operations makes a daily, self-organizing meeting, as used in Scrum, seem especially appropriate. Top-down and bottom-up methods merge naturally in centralized organizations; the author of this paper, as Director of Product Development, has the appropriate business-facing position to act as a kind of Scrum Master, hence his informal title of Queue Master, while product leads (answerable to him on the company's organization chart) are committed to prioritizing the Queue.

The main difference between IQ and Scrum is that the core of the development team, the LSI Technology Team, is continuously prioritizing the requirements relating to over a dozen major products. Each area of activity on each project must occupy the attention of a significant portion of the team within the time-box of all the others. Furthermore, change and customization is welcome between development stages. At the outset on a new project, development may be a single-stage, roughly defined Queue item. In the daily meeting, the team splits this whole into sub-tasks.

Alongside this, the team refashions the storyline that is said to timebox the larger project. When the LSI team splits a task into sub-tasks, each sub-task becomes an entry in the Queue. Each undergoes iteration. The integrity of each is safeguarded by the Queue Master during subsequent stages. The developing story grows into an epic, as development proceeds towards a larger combined or integrated project, such as new version release.

Programmers design, test, and build. Management can trigger or halt queue iteration, but must count on the originality and innovation of programmers to run the stage. The Queue Master watches over the procedure to ensure that daily meetings, business-driven changes, and the independence of the programmers are all maintained. The checks and balances between team-member roles work to prevent such faults as fragmentation of the development process, so-called "feature-creep," or disregard of larger business goals. In essence, stage-based time-boxing unleashes the creativity seen in the naïve "code-and-fix" mode, systematizes its relation to the business as a whole as activity-oriented methods did, but forestalls focuses away from process (feature-completion or documentation) to product and the changing needs of the user.

Most significantly, for LSI, it is a process that is transparent to end-users who have very precise, individualized needs. Consider the following case:

A school with multiple campuses has internal reporting as part of a quality enhancement program (QEP) in support of accreditation and separately reports the performance of each campus. The school needs usage statistics presented by campus rather than in block form. A quick conference with management leads to a new queue item specifying a per-campus report module. The Technology team (headed up by CTO Dr. Yanmu Zhou) dissects this into database- and user interface-related portions. Both are time-boxed for the next development stage.

Developers build test-scripts, and communication with the school's campus IT officials follows to determine proper parameters. LSI operations teams get involved; if it is an online tutoring integration, the Director of Online Tutoring would work with developers and user contacts to clarify issues of presentation.

By the close of the following development stage, LSI Quality Assurance, headed up by the Director of Operations, will confirm functionality of finished (and already tested) code. On the basis of this, a queue item may even be created that will bring this functionality into the next update of NetTutor, the LSI online tutoring service.

In any case, the next iteration of the queue will contain an item on improving usability and fine-tuning the internals of the specialized per-campus reporting module. Assuming that this new feature is judged by the entire development team to be worth inclusion for the next update, creation of the update is already underway.

The users, such as educators at the multi-campus school above, become vital critics of the new feature. In this sense, they are a part of the extended development team. If additional issues arise during the development of per-campus statistics presentation, this informs not only LSI but also the campus.

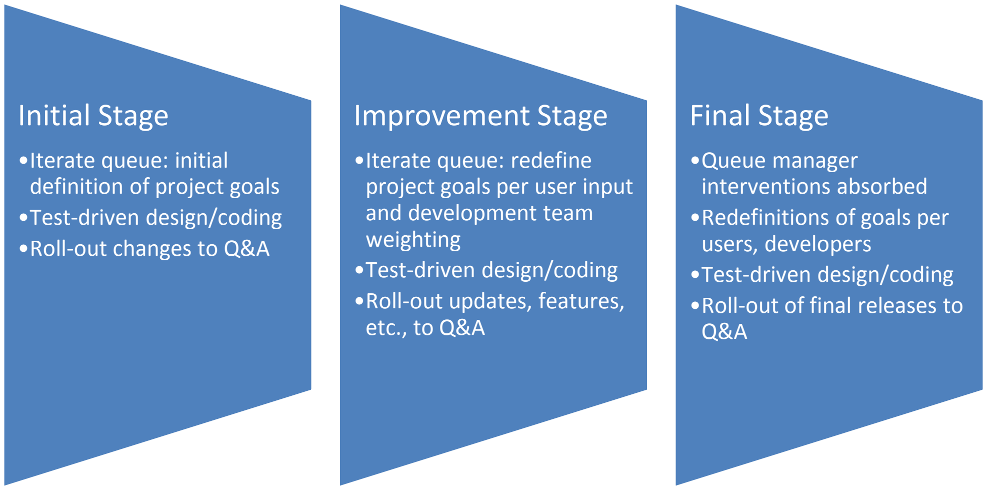
Initial Stage
•Iterate queue: initial definition of project goals
•Test-driven design/coding
•Roll-out changes to Q&A
Improvement Stage
•Iterate queue: redefine project goals per user input and development team weighting
•Test-driven design/coding
•Roll-out updates, features, etc., to Q&A
Final Stage
•Queue manager interventions absorbed
•Redefinitions of goals per users, developers
•Test-driven design/coding
•Roll-out of final releases to Q&A

Figure 2. Agile IQ Development: Typical Types of Development Stages

Using Agile IQ, management has full discretion to announce product version releases, so long as the announcement is made between development stages, without impairing the usefulness of the new version. The Queue items involved in bringing out the new release can be separately announced as they are implemented. The new release would itself go onto the Queue for later iteration and updates. In this way, Agile IQ is a procedure, like Scrum, with continuous improvement built into the process.

The LSI Marketing team works as part of this process to acquaint the team as a whole with the requirements of potential users of LSI products. Management then sets the customer-participant component into motion by placing user requirements, as new features, on the development queue. The Queue Master ensures these are properly time-boxed in the daily development meetings so that new features are prioritized, do not overwhelm the functionality of a product, and so forth, so that the IQ process works to encourage the independent effort of programmers.

Product owners at LSI play a role that is both customer-facing and project managerial in character. Organizationally, the product owner answers to the Director of Product Development, participates in the daily Agile IQ meetings, and liaises with operations staff. Indeed, operations personnel play a significant role as well: they produce and edit non-technical and technical documentation of products.

Implications of the Choice of Software Development Procedure for Online Education

Software and Learning

The story of the increasing importance of technology for education and educational institutions is short and clear. The transformation of the "Computer Aided Learning" in the 1960s and 70s via the Internet-enabled online academic collaboration that blossomed in the 1980s, into a wide Web and Web-applications in educational institutions has also meant the transformation of educators' approaches, attitudes of learners, and assumptions among the public about what is needed and what is not needed in schools.

In particular, the sheer cost of ensuring a level playing field for all who want to learn would be unsupportable in traditional terms. Anyone who doubts this should consider where we would be without the reliable storage of indefinite quantities of student data without computer technology. In the past decade, previously reluctant educators now contemplate seriously what this change means for learning methodology or pedagogy.

The current generation of students–the so-called "millenials"–never knew a world in which information could not be obtained at the push of a mouse-button from a laptop or a touch on a smartphone screen. It follows that software development is about reaching today's students in the most engaging and cognitively effective manner.

In the early days of the firm, LSI discussions with educators turned on *whether* online collaborative learning was feasible. Today, the discussion is almost solely about *how* to select online tools and technologies, *how* to extend an institution's virtual footprint, and *how* to integrate software that best facilitates learning.

Nonetheless, in the development of LSI as a software firm and in all its development, guidance comes from the constructivist approach to learning – the idea that knowledge is constructed by the student through a structured experience (Vygotsky, 1962). Software intelligently organizes learning and uses its hierarchical relationship of learning goals (Anderson, 2001).

Putting the educator in charge: IQ, agility, and the effectiveness of tools

It is not accidental that Agile IQ and similar agile approaches are found in major learning firms. At LSI, this is the answer to how to bring the practicing educator into development while still relying on the creativity of a proven development team.

A famous postulate of agility is that change is the only constant. Agile IQ, with its relatively short development cycles, its reiterative reconsideration and redefinition of tasks, and its direct incorporation of the academic community in the process, is driven by the changes in education. Founding members of the LSI development team come from the academic world and were largely trained in earlier periods of learning software development. This is why the adoption of Agile IQ has been a conscious process and why LSI can assist its partners who are undergoing similar transformations (Schwaber, Agile Project Management With Scrum, 2009).

Whether in operations, product design, or supporting services like online tutoring, software development takes place in defined stages. At the end of each stage, the participants meet to gather thoughts about what has been accomplished. Team members also initiate research endeavors inquiring into such standard measures of learning success as retention and achievement.

In many cases, educators who want to adopt engaging technology have been obliged to repurpose social networking, podcasting equipment, and the like originally designed for the consumption of entertainment or news, since that was the software available. As charming and effective as some of these adaptations have been, they are also typically non-scalable and invite security problems (Menon, 2010). The vision of LSI in supporting distance learning turns this approach on its head. At LSI, agile software development supports the creation of software products which, by conscious design, are built to meet the needs of educators and rebuilt and customized as necessary to help achieve student success.

Agile IQ and Preparing the Future

What this means

There are several questions that anyone seeking to partner with an educational software firm should consider:

- What kind of business and professional track record does the contemplated partner have?
- Is software development based on a proven model?
- Are the products subject to clear standards easily understood in the academic community?

The LSI Agile IQ software development procedure makes the firm one of very few with a thought-out approach to development reflecting the aims of educational technology. As a business closely allied with the academic community, LSI has found three major benefits of adopting Agile IQ:

1) **Dependability is built into LSI software, not an add-on.**

   Consumers are familiar with after-market service contracts and the associated fees. These are not part of LSI services: Agile IQ starts from the premise that if it meets the needs of the user, it must be built into the service or product. One example of this is the extensive use of evolving "Rules of Engagement" to dynamically align service provided with user expectations. Anything expected from the service becomes part of the Agile IQ epic, enters the programming queue, and is timeboxed for on-time delivery. Iteration of the development queue pivots on a deeper awareness and articulation by the LSI development team of the concerns of the academic community; each development stage marks concrete improvements in LSI's capacity to support distance and mobile learning.

2) **There is someone for IT to talk to when difficulties arise.**

   Since the user adopting LSI software is brought into the software development cycle on equal footing with its designers, this means real-time improvement and a continuing collaboration. It matters to LSI that specific pedagogical concerns are addressed successfully. In fact, the strength of development is thrown into connecting technology to learning goals and the goals of learning institutions.

3) **As technology advances, so does LSI software.**

   Web languages and protocols can mean improvements in the speed of access to the Internet. From one stage of development to the next at LSI, these go into recasting the story that determines what will be done. Professionals have grown used to the operation of Moore's law, namely, the observation that the number of transistors on a piece of silicon doubles every eighteen months. This also means that the best way to code for Web-based learning undergoes significant change at two-year intervals. Agile IQ, with its timebox of Queue iterations, enables LSI to put increased digital power at the service of educators.

Summary

For those who want to understand more about how Agile IQ works, LSI welcomes inquiries. Just as with each of its software products and services, the developers at LSI look for user and professional feedback. In fact, Agile IQ relies upon the engagement of users.

Visits, webinars, and participation in professional meetings are all part of the LSI agenda. Any company seeking a service provided by LSI, any school or program seeking to support its online outreach with online collaboration, online tutoring, online content and assessment, or data visibility would do well to start with considering the results of Agile IQ software development procedure. There is scholarly research available to support the effectiveness of the products. The practice of Agile IQ ensures that as the world of education evolves, LSI continues to bring out the features, products, and platforms suited to the changing landscape.

Ultimately, LSI looks to the experience of educators for its inspiration in development. Agile IQ aligns the development process with the richness and diversity of that experience.

Appendix I: From the *Agile Manifesto*

Principles behind the Agile Manifesto

We follow these principles:
Our highest priority is to satisfy the customer
through early and continuous delivery
of valuable software.

Welcome changing requirements, even late in
development. Agile processes harness change for
the customer's competitive advantage.

Deliver working software frequently, from a
couple of weeks to a couple of months, with a
preference to the shorter timescale.

Business people and developers must work
together daily throughout the project.

Build projects around motivated individuals.
Give them the environment and support they need,
and trust them to get the job done.

The most efficient and effective method of
conveying information to and within a development
team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development.
The sponsors, developers, and users should be able
to maintain a constant pace indefinitely.

Continuous attention to technical excellence
and good design enhances agility.

Simplicity--the art of maximizing the amount
of work not done--is essential.

The best architectures, requirements, and designs
emerge from self-organizing teams.

At regular intervals, the team reflects on how
to become more effective, then tunes and adjusts
its behavior accordingly.

(Principles behind the Agile Manifesto)

Appendix II: Scrum Principles

The framework and terminology are simple in concept yet difficult to implement. Successful Scrum teams embrace the values upon which Scrum is based (paraphrased from the Agile Manifesto):

We value

Individuals and interactions over processes and tools

Completed functionality over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, the items on the left matter more.

True success with the Scrum framework comes from teams and organizations who understand these values and the principles that form the foundation of all agile processes.

A Few Detailed Definitions

Product backlog: A product backlog is dynamic—Items may be deleted or added at any time during the project. It is prioritized—Items with the highest priority are completed first. It is progressively refined—Lower priority items are intentionally coarse-grained.

Sprint backlog: A sprint backlog is a negotiated set of items from the product backlog that a team commits to complete during the timebox of a sprint. Items in the sprint backlog are broken into detailed tasks for the team members to complete. The team works collaboratively to complete the items in the sprint backlog, meeting each day (during a daily scrum) to share struggles and progress and update the sprint backlog and burndown chart accordingly.

Scrum Terminology

We've introduced some new terms in describing the Scrum framework. Let's look at them in more detail. Scrum is made up of three roles, four ceremonies, and three artifacts.

Three roles

- Product owner: responsible for the business value of the project
- ScrumMaster: ensures that the team is functional and productive
- Team: self-organizes to get the work done

Four ceremonies

- Sprint planning: the team meets with the product owner to choose a set of work to deliver during a sprint
- Daily scrum: the team meets each day to share struggles and progress
- Sprint reviews: the team demonstrates to the product owner what it has completed during the sprint

- Sprint retrospectives: the team looks for ways to improve the product and the process.

Three artifacts

- Product backlog: ordered list of ideas for the product
- Sprint backlog: set of work from the product backlog that the team agrees to complete in a sprint, broken into tasks
- Product Increment: required result of every sprint. It is an integrated version of the product, kept at high enough quality to be shippable.

(Principles of Scrum)

Works Cited

Anderson, L. W. (Ed.). (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives.* Boston, MA: Allyn & Bacon.

Berk, L., & Winsler, A. (1995). *Scaffolding Children's Learning: Vygotsky and Early Childhood Education.* Washington, DC: National Association for the Education of Young Children.

Cohn, M. (2010). *Succeeding with Agile: Software Development Using Scrum.* Ann Arbor, MI: Pearson Education.

Fu, W.-T., & Anderson, J. R. (2006). From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology: General, 135*(2), 184-206.

Kersaint, G., Barber, J., Dogbey, J., & Kephart, D. (2011, February). The Effect of Access to an Online Tutorial Service on College Algebra Student Outcomes. *Mentoring and Tutoring: Partnership in Learning*.

Kim, P., Miranda, T., & Olaciregui, C. (2008, July). Pocket School: Exploring Mobile Technology as a Sustainable Literacy Education Option for Underserved Indigenous Children in Latin America. *International Journal of Educational Development, 28*(4), 435-445.

Krathwohl, D. (Autumn 2002). Revising Bloom's Taxonomy. *Theory Into Practice, 41*(4), 212-218.

Menon, S. (2010). A Pedagogy/Andragogy-Neutral Learning Platform for Improving Effectiveness of Online Learning. *Proceedings of the Sixth Pan-Commonwealth Conference.* Kochi, India: In Press.

Piaget, J. (1998). *De la pedagogie.* Paris: Odile Jacob.

Principles behind the Agile Manifesto. (n.d.). Retrieved December 2012, from Agile Alliance: http://agilemanifesto.org/principles.htm

Principles of Scrum. (n.d.). Retrieved December 13, 2012, from Scrum Alliance: http://www.scrumalliance.org/pages/scrum_101

Schwaber, K. (2009). *Agile Project Management With Scrum.* O'Reilly.

Schwaber, K., & Beedle, M. (2001). *Agile Software Development With Scrum.* Prentice Hall.

Smith, G., & Klein, W. (2004). Diagrams and math notation in e-learning. *International Journal of Mathematics Education in Science and Technology, 35*(5), 681-695.

Vygotsky, L. (1962). *Thought and Language.* Cambridge, MA: MIT Press.

About Link-Systems International, Inc.

Link-Systems International, Inc. (LSI) is a market-leading software company that helps academic institutions create effective digital solutions for teaching and learning. Many of the best-known brands in education use our web-based platforms:

- NetTutor® – our online tutoring solution
- MyAcademicWorkshop™ – our adaptive placement, assessment, and homework system for mathematics
- WorldWideWhiteboard® – our mobile-ready collaboration suite
- Information Visibility Solutions™ – our enterprise analytics dashboards for education

Research-Based Approach

LSI employs Integrated Cognitive-Contextualized Learning (ICCL) in order to deliver research-proven online learning solutions that are based on the latest developments in learning theory. Educators work hard to engage students and to help them reach their learning goals; we offer theory and practice to supply educational technology in support of that work. LSI has recognized the unity of two important arms of constructivist learning theory—the measurement of self-achieved cognitive advances against agreed-upon standards and the contextualization of learning to create learning initiatives for the active learner.

Over the years, LSI has accumulated a wealth of experience with educators seeking new ways to engage their students through unique and involving learning experiences and measure and reinforce concrete cognitive advances. According to ICCL, these are simply two sides of the same experience: implementation of online learning tools enables both the delivery of contextualized content and the verification of learning achievement.

The application of ICCL has practical results for LSI, whose solutions are all based on the theory. LSI uses ICCL to develop solutions that meet the specific needs of its learning institutions and their partners. At the same time, these solutions make educational achievement measurable to ensure learning success while peer-reviewed research supports their effectiveness. The partnerships between educators and LSI are helping to define the new face of quality learning in the digital age.

Our Company

Launched in 1995, LSI has created several unique and powerful software platforms that facilitate the sharing of content over the Internet. We specialize in mathematics, technical and scientific content, the most critical types of online content with respect to student persistence, and the most difficult to share online.

Our customers include K-12 publishers, higher education publishers, virtual high schools, higher education institutions, technology companies, and joint programs dedicated to providing online educational content to members of organized labor and their families.

We are located in Tampa, Florida, a few miles from the University of South Florida, which has excellent engineering, computer science and mathematics programs, providing LSI many of its employees.

Student Persistence and Retention

Today, LSI is recognized by a variety of publishers and educational institutions not only for its high-quality work and dedication to meeting commitments, but also for its unique ability to develop digital solutions that are tailored to the needs of its customers.

Our partners and customers have come to value and trust LSI because we are the only company that offers a complete suite of interoperable solutions that address the entire life cycle of the student, with an overt focus on the bottom line: student persistence and student retention.

LSI Mission Statement

Link-Systems International is the leader in providing integrated technology and service solutions to educators in order to improve the quality of education and training, ensure student success and retention, and provide affordable education to students, workers, and their families.

Corporate Executive Team

Vincent T. Forese, President, Chief Executive Officer
William K. Barter, Senior Vice President, Sales, Marketing, and Business Development
Dr. Emil Moskona, Senior Vice President, Chief Operating Officer
Dr. Yanmu Zhou, Senior Vice President, Chief Technology Officer
Dr. Milena Moskova, Vice President, Research and Development

About Academic Research at LSI

We are enthusiastic about the commitment of institutions and academics to the use of technology with proven benefits to their students. If you would like to write about the impact of Web-based technology, please let us know. We encourage educational research and will work with you and your staff to develop scientific studies into the relationship of the online learning experience to successful student outcomes. Please contact our Academic Research Department.

David Kephart, PhD
Director of Academic Research
Link-Systems International, Inc.
4515 George Rd., Suite 340
Tampa, Florida 33634
(813) 674-0660 x207
dkephart@link-systems.com