ADDIE and Agile: How ADDIE Survives in an Agile World

Jennifer H. Wolfe

SLD 550 – Strategic Thinking, Analysis, Planning, and Evaluation: The Capstone

University of Indianapolis

Abstract

This paper compares and contrasts the Successive Approximation Model (SAM) instructional systems design (ISD) approach, with the ADDIE framework, and demonstrates that though ADDIE is traditionally executed in a very Waterfall manner, it does not have to be.

Two practical approaches to using ADDIE on Agile Scrum projects are provided. The first shows how the instructional design and development work can be incorporated into the sprint itself. The second uses a separate team to do the instructional design and development work following the completion of the sprint. Both demonstrate how ADDIE survives in an Agile world.

Keywords: ADDIE, Agile, Scrum, Instructional Design, Instructional Systems Design, ISD, SDLC, Systems Development Life Cycle, Kirkpatrick, Levels of Evaluation, Successive Approximation Model, SAM

Background

The concept of ADDIE for instructional systems development (ISD), also known as instructional design (ID), has been around since the 1970s; but its origin is uncertain. Molenda in his article, "In Search of the Elusive ADDIE Model" (2003), states that based upon his research, the label "ADDIE Model" appears to "not have a single author, but rather to have evolved informally through oral tradition" (p.1).

The general understanding is that ADDIE is an acronym for Analysis, Design, Development, Implementation, and Evaluation, and that these phases are sequential and iterative (see *Figure 1*.) Beyond that any claims about what ADDIE is, or is not, are the invention of the author (Molenda, 2003).

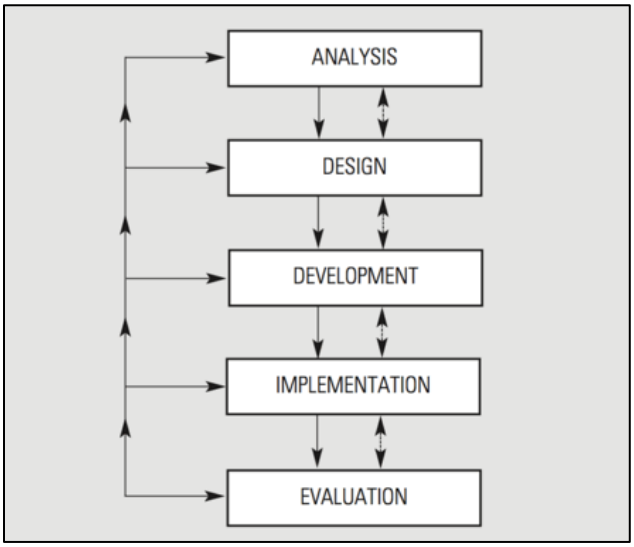


Figure 1. ADDIE Diagram. Reprinted from "The ADDIE Model" by C. Hodell, 2005, *Basics of Instructional Systems Development,* INFO-LINE, Issue 9706, p. 3. Copyright 2005 by American Society of Training and Development.

Bichelmeyer (2004) points out that ADDIE is actually a conceptual framework that loosely guides instructional designers to approach instructional design in a systematic way. Whereas a model is "a representation that accurately resembles an existing structure" (p. 4), which is not the case with ADDIE. It is also not a methodology because they are even more

prescriptive, by providing detailed step-by-step procedures to follow. Therefore, for the rest of this paper the term "ADDIE framework" will be used, not "ADDIE Model."

Recently several sources have challenged the suitability of the ADDIE framework due to the rise in popularity of Agile system development methodologies. These sources suggest their own models work better, and that ADDIE will not work in an Agile world. This paper challenges that assumption and instead provides two potential approaches to how the ADDIE framework could be adapted to Agile projects without the need to buy new books or hire consulting services.

This paper begins with an overview of the fundamentals of ADDIE and Agile. Next, it reviews a new alternative ISD model called Successive Approximation Model (SAM) and then compares and contrasts it with ADDIE. Lastly, two potential approaches to combining ADDIE and Agile projects are reviewed.

ADDIE can be used for any kind of ISD or instructional design project; however, this paper uses examples from the field of information technology (IT) as a case study. This is appropriate since Agile is primarily an IT software development methodology and most Agile models do not address how or when training on the new system should be developed. This is not unique to Agile projects, as training is often overlooked on Waterfall IT projects as well.

Systems Development Life Cycle (SDLC)

The term Systems Development Life Cycle (SDLC) is used to describe the processes that IT project teams utilize to develop a new IT system, including new hardware or software configurations. SDLC is a framework similar to the ADDIE framework in that it too is a basis for many SDLC models, including Agile Scrum that will be described later. All of the SDLC models include some version of the following processes: Feasibility Analysis and Planning, Requirements Gathering and Analysis, System Design, System Coding (Development), Quality

Assurance Testing, Implementation, and Maintenance. *Figure 2* is diagram of a standard SDLC framework.

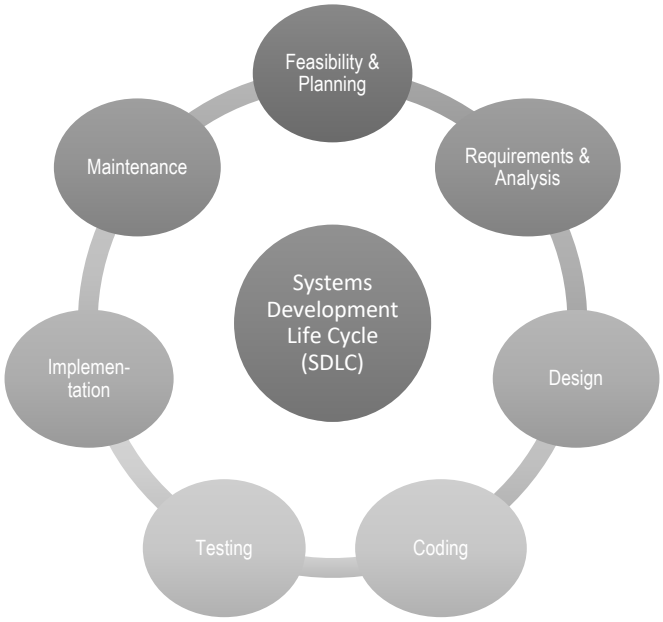


Figure 2. Standard SDLC Framework

Both the ADDIE and SDLC frameworks are fundamentally about developing something (training or a new system, respectively) based upon the stakeholders' needs. Regardless of what you are developing, how you group or separate the steps, or the names you give the processes, both frameworks fundamentally go through similar phases (see *Figure 3*).

SDLC	Feasibility & Planning	Requirements & Analysis	Design	Coding	Testing	Implemen-tation	Maintenance
ADDIE		Analysis	Design	Development	Implementation		Evaluation

Figure 3. SDLC and ADDIE Frameworks Combined

Ideal Instructional Design Methodology

The goal of any instructional design and development project is to develop effective training. But what does that mean? According to Allen (2012), effective training is (p. 22-23):

1. **Meaningful** – It connects the new information being trained to information the audience already knows. It is clear about what the participants should be able to do at the end of the learning event.
2. **Memorable** – It utilizes industry best practices to increase retention, including delivering the training close to the time it will be used, using hands on exercises, repetition, evaluation, and mnemonics as examples.
3. **Motivational** –It incorporates real life examples and scenarios that help adults learn. Also it includes why the topic is important and builds on existing motivation.
4. **Measureable** – It produces skills that can be observed and provides time to practice and receive feedback on performance.

Training created using any framework or model should exhibit these features. Additionally, according to Allen (2012), the ideal instructional design process should also be (p. 29-33):

- **Collaborative** – Involve the learners and subject matter experts (SMEs) early and often to get their feedback when it can make the greatest impact.
- **Iterative** – Gather frequent stakeholder feedback beginning early in the process when changes have the least impact and therefore are the cheapest to implement.
- **Efficient and Effective** – Factor in constraints while producing the best quality learning possible. Expect and plan for changes that may cause re-work.

- **Manageable** - Provide clear direction for team members so that everyone knows what needs to be done and who is doing it.

Allen's (2012) view of ADDIE is very rigid, inflexible, and moves in only one direction. It is slow and does not assure quality. However, as Molenda (2003) states any additional qualities or features beyond the five phases, their sequence, and iterative nature, are not inherent in the ADDIE framework itself. The next section examines the framework in more detail and includes an evaluation of how well it addresses the quality criteria introduced above.

The ADDIE Framework

The five phases of the ADDIE framework are: Analysis, Design, Development, Implementation, and Evaluation. A brief overview of each phase and how they can be used on an IT implementation project are included below. The examples are not prescriptive but are used for demonstration purposes since ADDIE is a framework and not a methodology.

Analysis

The first phase of ADDIE is Analysis and its goal is to identify the needs of the users and the topics they need to learn. This is where instructional designers gather all available information about the current and future states of the IT systems and business processes impacted by the changes. Information about the impacted audience, the business culture, available training delivery methods and systems, and budget are all important because they impact the potential design.

Typically, on an IT project the output of this phase is a list of proposed courses each with a brief, high-level description, expected audience, and delivery method. This information and any assumptions made will be validated and potentially changed in the next phase. Iteration has

always been part of ADDIE. As you learn, you adapt. This is also a fundamental principle of Agile as well, which we will discuss in the next section.

The Analysis phase corresponds to the Requirements & Analysis phase in the standard System Development Life Cycle (SDLC) framework introduced earlier. This is where the project team gathers the users' system requirements and analyzes them to determine the needs and impacts on the existing infrastructure and software, as well as determining what work they need to perform, just like the instructional designers are doing with the training content.

Design

During the Design phase, the courses are designed. This typically means that a course outline is created for each course. The course outline is a form of training course blueprint. Much like a blueprint for a house, it brings to life the designer's vision, allowing stakeholders to provide input on that vision before construction begins, and changes become costlier. Some, but not all changes made AFTER the course (or house) are under construction are more difficult and costly to correct. However, if they are important they can always still be made and in many cases they can made with minimal impact to the budget or timeline.

It has become standard that the course and the individual modules within the course all have learning objectives. Learning objectives set expectations and provide clear, measurable criteria on which to evaluate the quality of the training materials (do the materials cover the objectives sufficiently) and the expected performance of the participants at the end of the learning event (can the users do what they have been taught).

Sometimes prototypes are developed to allow stakeholders to see samples of what the training deliverables will look like. If more than one course will be created, a template should be

built from the approved prototype, especially if more than one instructional designer will be creating the final deliverables. This will improve consistency and quality between designers.

The ADDIE Design phase corresponds to the traditional SDLC Design phase where the development team is creating prototypes and design documents for review and approval before they begin developing the new system. This is similar to the way instructional designers get approval before developing the training materials.

Development

The Development phase is where the course outline (a course design document) is used as a blueprint to build the actual course materials. If a prototype or template were created, they should be used as a starting point for the development of all actual deliverables. This not only increases the overall quality of the training, but saves valuable time in the long run. Novice training developers often jump in and start with development, not understanding the importance of the prep work in the first two phases.

While the system development team is building the new system, the instructional designers can begin developing the training materials. In some cases, the Development phase is extremely long or a separate team may be engaged to develop the materials based upon the course outlines.

Either way, the development of the training materials must begin before the system is completely built. Most project timelines do not allow for the training development to begin AFTER the system development is complete and fully tested. Therefore, it is important for the instructional designers to work closely with the system development project manager to identify which parts of the new system, or processes are complete, defect-free, and unlikely to change, so that the instructional designer can begin their work in these areas first and follow closely behind

the development team as they complete additional functionality. Occasionally, it might be necessary to replace screen shots or update processes, but if the two teams work closely together, rework can be minimized.

Implementation

The Implementation phase is where the course is delivered to an audience. It is strongly recommended to do a "pilot" of each course with a sample audience including SMEs, stakeholders, and instructional designers present to take notes on edits, confirm timing, etc.

Depending upon the size and structure of the organization, the instructional designer often transitions the course to trainers, who will train the end users. This process of preparing the trainers is called a "train-the-trainer." Often during the train-the-trainer or pilot, potential edits are found. This is another chance to improve. Effort should be made to apply these edits before the course is shared with the broader audience.

Typically, on a systems development project, while the training developers are piloting, performing train-the-trainer, or other activities, the system is continuing to be tested. The length of the Development and Testing phases will impact the exact timing of the training pilot and train-the-trainer sessions. This should be negotiated and determined at the beginning of the project during the planning phase.

The type of system being developed will also impact when end user training is conducted, either before or after the new system goes live in production. This corresponds to the Implementation phase of SDLC.

Evaluation

Typically, the Evaluation phase refers to the formal evaluation of the course and not the reviews conducted by SMEs that are conducted throughout the previous phases. The most popular evaluation model is Kirkpatrick's four levels of evaluation (Kirkpatrick, 2016).

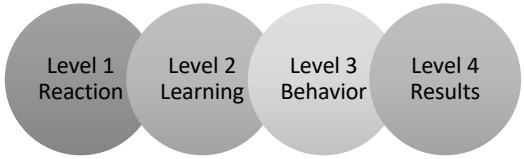


Figure 4. Kirkpatrick's Four Levels of Evaluation

Below is a description of each evaluation level:

- **Level 1 (Reaction**) – Sometimes called a "smile sheet," it asks: "What is the participant's initial reaction to the training? Are they satisfied? Were they comfortable? Was the trainer professional?"
- **Level 2 (Learning)** – Evaluation of the participants' learning at the end of the module or course that asks: "Did they learn what was expected? Can they answer questions on the content? Can they perform the tasks and processes in a sample environment?"
- **Level 3 (Behavior)** – After the participant returns to their job, it asks: "Are they applying what they learned in class?"
- **Level 4 (Results)** – After some time has passed and data is gathered, it asks: "Has the organization achieved the results they expected as a result of the training?"

Level 1 and Level 2 evaluations are commonly done on most system development training projects. Level 3 and Level 4 evaluations require sometime to pass after training is complete and therefore increase the post-production costs of a project. Typically, with IT projects the organization is anxious to get started on the development of the next new system, and little time and effort is spent after implementation to gather this kind of data. However, much

could be learned that could positively impact future projects, so these evaluations are strongly encouraged.

After Implementation, the next SDLC phase is Maintenance where the IT organization responds to change requests, system errors, etc. This would occur at the same time as post-training evaluation.

ADDIE - An Ideal Instructional Design Process?

Earlier we introduced the criteria that Allen (2012) recommends are part of an ideal instructional design process. Does the ADDIE framework meet these criteria?

- **Is it Collaborative?** – Yes, it can be. Stakeholder feedback can and should be gathered during every phase. There is nothing in the ADDIE framework that prohibits or discourages this.
- **Is it Iterative?** – Yes. According to Molenda (2003, p.1), it has always been iterative.
- **Is it Efficient and Effective?** Maybe, but no more or less than any other system. This depends on how it is executed. Any framework or model can be effectively or ineffectively executed. Using experienced instructional designers will help provide the most efficient and effective results. ADDIE is easy to remember and intuitive so the simplicity may help people learn, remember and eventually apply the model. This is merely a hypothesis.
- **Is it Manageable?** Yes. Even Allen (2012, p.32) concedes that ADDIE is easy to understand and manage. How well it is managed depends on the experience and skills of the team members and project manager.

Therefore, even though ADDIE is a framework and not a process, model or methodology, it has the capability of being an ideal framework for instructional design… if executed

appropriately. Any framework, model, etc. is only as good as how it is executed and the more experience the team has, typically the better the results.

Agile Software Development

The need to deliver software to stakeholders faster has been around since software was invented. Incremental software development methods can be traced back to the 1950s. But the Agile software development movement didn't gain much traction until 2001 when 17 software developers met at a resort in Utah to discuss lightweight development methods. There they developed the Manifesto for Agile Software Development, also called the "Agile Manifesto" (Highsmith, 2001). The complete manifesto is printed with permission in its entirety in the next section as per the guidelines.

Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

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

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

Agile Principles

The Agile Manifesto is based on twelve principles (Highsmith, 2001):

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (in weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Agile Scrum

As you can see from the Agile Manifesto and Twelve Principles, Agile is, at its core, about values and culture. In order for project teams to put these values into action, a model was needed. That is where Agile Scrum came in. It is founded on the concept that knowledge comes from experience and making decisions based on what is known. Scrum is an iterative, incremental approach to maximizing predictably and controlling risk (Schwaber & Sutherland, 2014).

According to Schwaber & Sutherland (2014), Scrum teams are self-organizing and cross-functional, and include all of the skills needed to complete their work without depending on

others outside of the team. They deliver products incrementally (in small slices of functionality), and iteratively, maximizing the opportunities to get feedback from the customer. This incremental delivery of "done" product ensures that a version of working product is always available which is typically not the case on Waterfall projects.

Scrum Team

Schwaber & Sutherland (2014, p. 4-7) define a Scrum Team as consisting of the following roles:

- **Product Owner –** Single person responsible for "maximizing the value of the product and work of the development team" (p.5). They manage the prioritized list of items for the development team to work on.
- **Development Team** – A self-organizing, cross-functional group that does the work to deliver potentially releasable "done" product at the end of each sprint. There are no titles on this team; everyone is a Developer.
- **Scrum Master** – Responsible for making sure that Scrum is understood and followed. They help the team make sure to adhere to Scrum practices.

Scrum Events

Recommended scrum events, also called "ceremonies," are time boxed activities used to create regularity, and to minimize the need for extra meetings. Below are the most common scrum ceremonies (Schwaber & Sutherland, 2014):

- **Sprint** – A two- to four-week time boxed event in which a usable, potentially releasable product increment is created and is "Done." It's best if sprint durations are consistent from sprint to sprint, and a new sprint starts immediately after the previous sprint ends.

- **Sprint Planning –** A time boxed event of no more than eight hours per sprint. During sprint planning the development team learns what the product owner has listed as top priority in the product backlog and the team discusses what can be delivered and how it will be achieved (a form of Requirements Analysis). Only the development team can determine how much they can get done in the sprint.
- **Daily Standup or Scrum** – A fifteen-minute time boxed daily meeting to discuss the results of the last 24 hours and to coordinate activities for the next 24 hours. Three questions are asked:
    1. "What have I done since the last meeting (during the last 24 hours)?
    2. What am I going to do in the next 24 hours (before our next meeting)?
    3. Are there any impediments preventing me from meeting the sprint goal?"
- **Sprint Review or Demo** – Event held at the end of the sprint to demonstrate to the Product Owner and any other stakeholders what was "done" during the sprint.
- **Sprint Retrospective** – An event that occurs at the end of the sprint after the demo, but before the next sprint planning event. It is an opportunity for the Scrum team to evaluate itself and create a plan for improvement.

Scrum Artifacts

Below are the artifacts that are used in the Scrum events (Schwaber & Sutherland, 2014):

- **Product Backlog –** An ordered list of what the Product Owner thinks is needed. The Product Owner is responsible for the product backlog.
- **Sprint Backlog –** A set of product backlog items that have been selected to be worked in in the sprint, in order to achieve the product increment and achieve the sprint goal.

- **Increment –** The sum of all of the product backlog items completed in a sprint. At the end of the sprint the new increment must be "Done."

An important part of setting the team up for success is agreeing on a "Definition of Done," which is a shared understanding of what it means to say an increment is "done." This should be discussed and agreed upon before the sprint begins. Also, just because a piece of functionality COULD be released does not mean that it automatically is. Often functionality is grouped into "Releases" so that the tool is not continually changing. Because that could be frustrating to users and make training challenging. Below is a diagram that shows how the roles, Artifacts and ceremonies, work together in Agile Scrum (Norex, 2016).

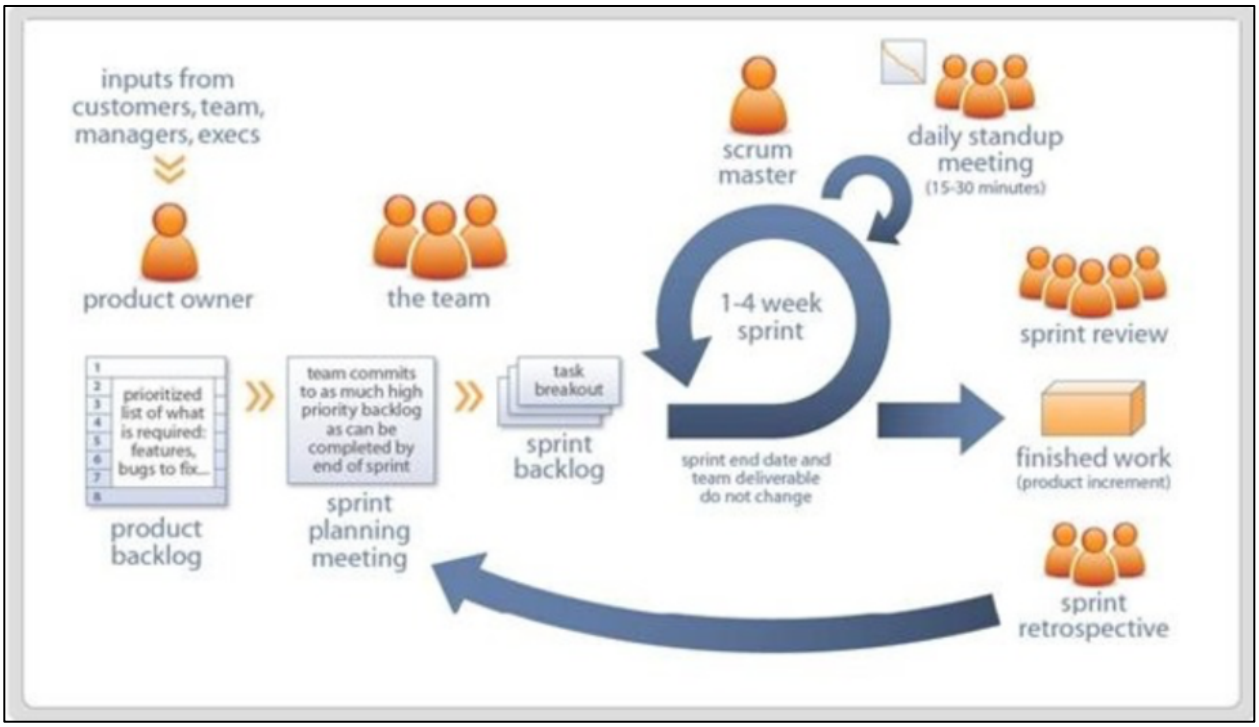


Figure 5. Agile Scrum Diagram including all ceremonies, artifacts, and roles. Reprinted from: Norex, Inc., 2016, *Agile Scrum Process.* Copyright 2016 by Norex, Inc.

All of the relevant components of SDLC that involve building a new system function (Requirements and Analysis in sprint planning and Design, Coding, and Testing in the sprints themselves) are all present in Agile Scrum because they are so fundamental to building as to be

unavoidable, as we suggested earlier. Even though the way the teams are formed, how the work is assigned and how it is prioritized is different in Agile Scrum than Waterfall, the fundamental SDLC framework is still present.

In an effort to get results faster and to serve their customers better, the authors of the Agile Manifesto and the Agile Scrum model did not decide to skip requirements gathering and design and just jump in and start coding. Quite the contrary. They have brought the customer into the team as a product owner so that they can get regular feedback to make sure they are building what the customer wants.

ADDIE is similar. It is a conceptual framework that is so fundamental to the way that training is built, that it too cannot be avoided. The next section reviews the Successive Approximation Model (SAM) and compares and contrasts it with the ADDIE framework.

Successive Approximation Model (SAM)

Allen (2012) developed the Successive Approximation Model (SAM) as a way to develop e-learning courses in an Agile manner, although it is proposed for developing any training content. There are two versions of SAM: SAM1 and SAM2.

Successive Approximation Model 1 (SAM1)

The basic SAM (SAM1) is fairly straight forward, see *Figure 6* below. You will recognize the terms in the boxes from the introduction to ADDIE. It is intended for small teams or individuals that do not require specialized skills (such as programming or video production).

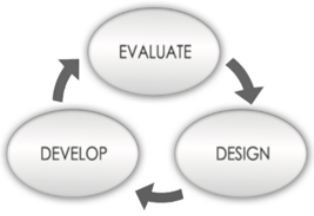


Figure 6. Overview of SAM1. Reprinted from Allen, 2012, *Leaving ADDIE for SAM.* Copyright 2012 by American Society of Training and Development.

Allen (2012) recommends repeating SAM1 three times starting and ending with Evaluate (the initial Analysis phase was renamed to Evaluate to accommodate the model's cyclical design). However, what is done in each iteration differs.

Iteration 1

1. **Evaluate:** Perform a quick analysis of the needs and goals.
2. **Design:** Prepare a rough design.
3. **Develop**: Prepare prototypes using simple tools that demonstrate the design.

Iteration 2

1. **Evaluate:** Review the Iteration 1 design with the stakeholders and identify what works and does not work.
2. **Design:** Sketch new alternatives.
3. **Develop:** Prepare prototypes that more closely resemble the final product.

Iteration 3

Iteration 3 is performed in a manner similar to Iteration 2, although each iteration should become more focused on development rather than design.

Successive Approximation Model 2 (SAM2)

There is also a SAM2 that includes three phases: Preparation, Iterative Design, and Iterative Development instead of repeating SAM1 three times (Allen, 2012). SAM2 is recommended for large, complex projects that have separate teams doing the design and development work, and is more descriptive about what happens in each iteration. As a result, it is more complex.

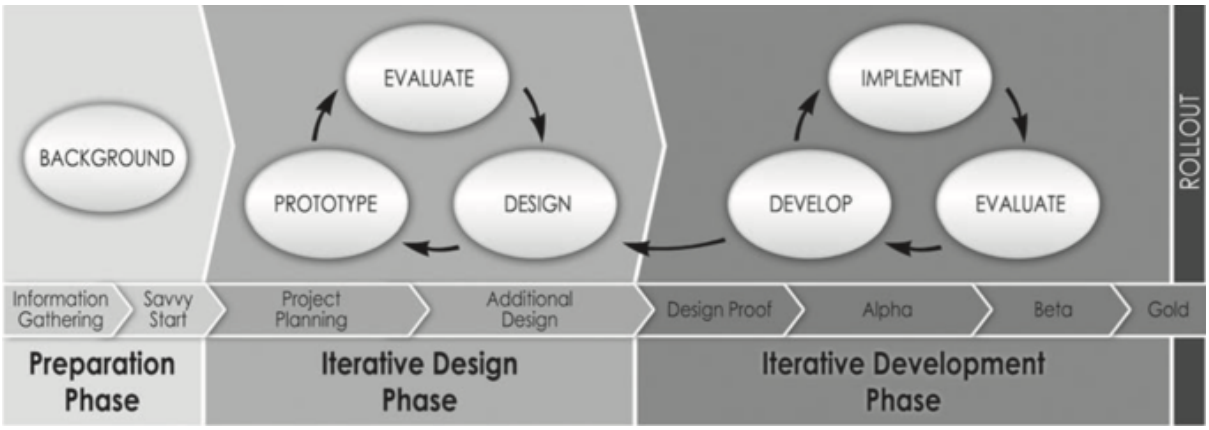


Figure 7. Overview of SAM2. Reprinted from Allen, 2012, *Leaving ADDIE for SAM.* Copyright 2012 by American Society of Training and Development.

Preparation Phase

The preparation phase includes gathering information on the background of the project, other similar projects and their success and challenges, samples of other training materials, project constraints, roles and responsibilities, and project goals (Allen, 2012). The SAM2 model also recommends a Savvy Start brainstorming session with the project team and stakeholders (including end users) to review the information gathered, and to generate design ideas. This ceremony (using an Agile Scrum term) will bring to light the true decision maker(s) and what outcomes are needed to demonstrate success (Allen, 2012).

In ADDIE terms this would be equivalent to the Analysis phase. In SDLC terms this would be the Requirements Analysis phase.

Iterative Design Phase

According to Allen (2012), for the project to be successful, Project Planning must be conducted during the Iterative Design phase to assess and communicate the budget, resource, and time needed to develop the product(s) designed during the Iterative Design phase. Good project planning is always iterative, with new information, details, and changes continually being incorporated into the plan, whether that plan is Waterfall or Agile.

At the end of the Iterative Design phase, prototypes of all of the training deliverables should have been designed, reviewed, and updated until the stakeholders are satisfied. Common prototypes include: Media (the look and feel), Functional (how they will work), and Integrated (combining the media and functional prototypes into one). This phase would correlate to the Design phase in both ADDIE and SDLC.

Iterative Development Phase

SAM2 differs from SAM1 because with SAM2, the project moves to the Iterative Development phase after the Iterative Design phase is complete. This is where the development is handed off to be completed by another team (contractors or technical programmers), if that is the labor model. The Iterative Development Phase involves gathering frequent stakeholder feedback as the content is developed (Allen, 2012). The first version that the stakeholders will see is called a Design Proof where the design prototype approved in the previous phase is brought to life with real content. It is basically a "visual, functional demonstration of the proposed solution that integrates samples of all components to test and to prove viability." During this process it may be discovered that additional design work is needed. If so, that piece of the functionality returns to the Iterative Design Phase.

The design proof is iterated until the stakeholders are satisfied. Then the Alpha version is created and shared with the stakeholders for feedback. Only minor errors are expected during this phase.

Any errors found in Alpha are fixed and the product is returned to the stakeholders for final approval, called Beta. Ideally, no errors will be found during the review/evaluation of the Beta and this version becomes the Gold version. If any errors are found the corrected version

becomes Beta 2, and so on until a clean version, called the Gold version, is achieved. The Gold version is rolled out and the SAM model is complete.

This correlates to the Development phase and the beginning of Implementation phase in ADDIE and SDLC, but SAM1 and SAM2 stop abruptly there. Allen (2012) used e-learning as a case study. Simply "rolling it out" to a Learning Management System might be sufficient for Implementation of an e-learning course, but for Instructor Led Training, there is often more to the Implementation phase including a pilot, train-the-trainer, logistics, etc. Also the Evaluation (ADDIE) and Maintenance (SDLC) of the training program post-implementation, such as the Kirkpatrick model introduced earlier, are not covered at all. Evaluate in SAM appears to only mean either Analysis (Pre-Design) or Pre-Development and Pre-Implementation content reviews.

ADDIE versus SAM

As you can see, there are similarities and differences between ADDIE and SAM. This section examines them in more detail, starting with the differences.

Differences between ADDIE and SAM

Below are a few of the differences between the two approaches:

1. ADDIE is a fundamental conceptual framework upon which all instructional systems design models and methodologies are built. SAM is an example of one of the models built from ADDIE.
2. ADDIE is a simple, easy-to-remember acronym that serves as mnemonic to remind instructional designers of the phases. SAM has two versions (SAM1 and SAM2) and is more complex and difficult to remember.

3. The initial Analysis phase of ADDIE has been renamed to Evaluate in SAM and includes Background, Information Gathering, Savvy Start, and Project Planning. All activities that would typically be included in Analysis.
4. SAM calls out the Evaluate step only within the Preparation, Design, and Development phases BEFORE the roll-out of the learning solution. SAM seems to use Evaluate as a form of review versus leverage the Kirkpatrick Levels of Evaluation discussed earlier and commonly used on training development projects.

Similarities between ADDIE and SAM

Even though ADDIE is a framework, SAM is a model, and Allen (2012) in his book title recommends "Leaving ADDIE for SAM," there are a remarkable number of similarities between the two approaches. Below is a list of a few of the most obvious examples:

1. Both are sequential. For example, it is not logical to switch the order of phases and begin with Develop and then go to Design. There is a logical flow from Preparation to Design and then Development in both approaches.
2. Both begin with steps to gather information and plan the work which ADDIE calls Analysis and SAM calls Evaluate.
3. Both have Design, Development, Implement, and Evaluate phases in that order.
4. Both are iterative, meaning that when information is gathered that changes an assumption made previously, you return to the required phase and make updates and then follow those updates through the other phases. Both indicate this with arrows.

Figure 8 overlays the ADDIE phases on the SAM2 diagram to show where they occur.

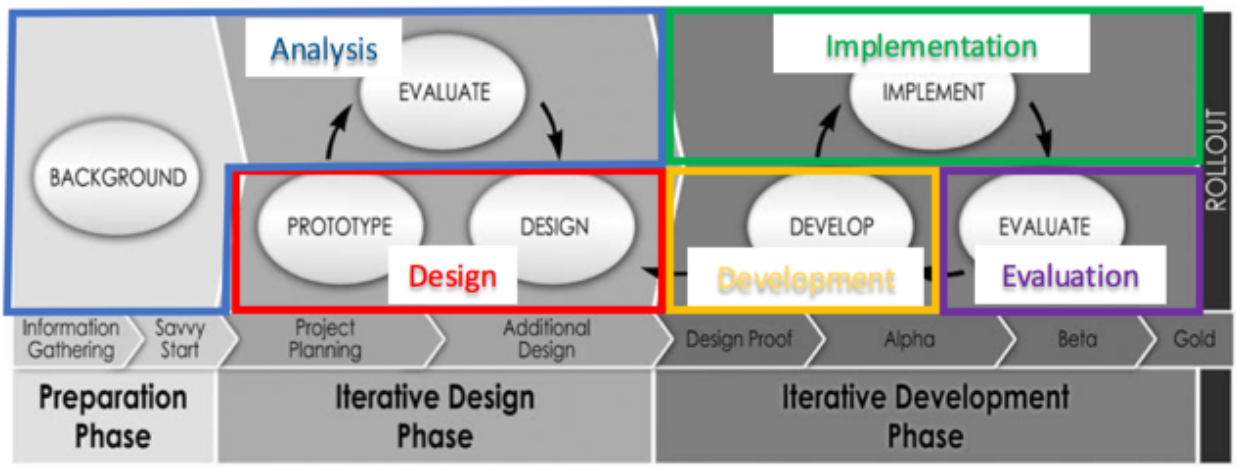


Figure 8. SAM2 and ADDIE. Adapted from Allen, 2012, *Leaving ADDIE for SAM.* Copyright 2012 by American Society of Training and Development.

Utilizing Agile principles such as involving the customer (end users) from the beginning of the process and welcoming changes, can improve the quality of the final instructional design product. These are great ideas. The point here is that these concepts are not unique to SAM nor counter to ADDIE. The only element of ADDIE that is not specifically called out in SAM is Analysis, yet Allen (2012) does use the term in the description of Evaluate, "Begin with a quick evaluation (analysis)..." (p. 34), and the following activities are clearly analytical: Background, Information Gathering, Savvy Start, and Evaluate. The Successive Approximation Model (SAM) while potentially useful for e-learning development, is fundamentally built from the ADDIE framework like the other ISD Models that Allen (2012) criticizes.

Proposal for Using ADDIE on Agile Scrum Systems Development Projects

So the ADDIE framework is alive and well, but can it fit into Agile Scrum so that development teams don't forget about developing training as well as new system functionality? It is possible and to help teams remember to not only build a tool that meets customer needs, but to also train those same users on how to use the new tool. Below the Agile Manifesto has been updated to incorporate training.

Agile Manifesto – Updated

- **Individuals and interactions** over processes and tools
- **Working software <u>& Effective Training</u>** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

In other words, instead of trying to develop the perfect instructional design model or process, we should be working with the development team, customers, end users, etc. to brainstorm, do peer reviews, share drafts, create, and share prototypes. Instead of gathering signoff sheets as cover, we should gather feedback from the stakeholders from the beginning, to make the training product feel like a collective work product. It shouldn't be about the big surprise or "reveal" at the end. Training isn't evaluated on its shock value; we want the final product to meet the criteria for effective learning, for the end users to learn and then change their behavior. In the case of new IT systems, we want them to start using the new functionality with minimal disruption to their lives.

All of these things are important and can be done with ADDIE. Below are two proposals on how that can be accomplished on an Agile Scrum software project.

Approach 1 – Training Development During the Sprint

In the first approach, the Analysis, Design, and Development work is completed by the development team as part of their sprint work and the training documentation is considered part of the Definition of Done. This can only happen when the development team has instructional design skills.

Because Agile development teams are usually developing new technical features for a system, the step-by-step procedures on how to use these new features needs to be documented.

They are called Work Instructions and are used to create testing scripts, for training and for production support for the future end users and IT Help Desk support. Tools like Oracle User Productivity Kit (UPK), Adobe Captivate, and Camtasia Studio can be used to record these steps. Usually these work instructions are used as part of a training course or larger training program and aren't expected to stand on their own. Therefore, the team should include someone with instructional design skills and experience to bring the entire training package together.

Also the time to do this work should be considered during sprint planning. That may mean that the product increment is smaller, or the team must complete their testing earlier in the sprint to allow time for the instructional materials to be built. Either way, the decision to include this documentation in the Definition of Done or not should be determined early on and planned accordingly.

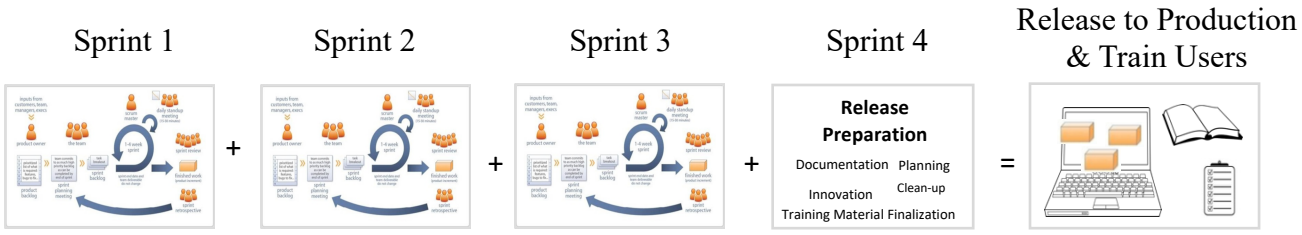
A decision must be made on when the potentially releasable products will be released into production. Typically this is not done at the end of every two-week sprint if there is user impact because it can be very disruptive. Many organizations have rules and processes on when and how new code is released into production, so often the potentially releasable products are grouped and released later based upon a corporate release schedule. Whenever the content is released, either at the end of the sprint or later based upon the release schedule, the training needs to be implemented at the same time. The Evaluation of the training should follow the Implementation. In summary:

- **Analysis, Design, Development** – Performed during the sprint as part of the Definition of Done.
- **Implement and Evaluate** – Conducted at the time of release to production and the users.

Approach 2 – Training Development After the Sprint

In the second approach the training development is not done by the development team as part of the sprint. It is done by extended team members following the end of the sprint. This approach is often used because the development team does not typically have instructional design skills.

The instructional designers would attend the sprint demo to learn about the new feature and functionality. They would then work on developing the training in the next sprint. They may work in an agile manner themselves or they may not be an Agile team at all. Either way, the development team would be working ahead of the instructional designers and sharing the new functionality they build each sprint during the sprint demo. Below is a diagram that depicts an example of this approach:



SDLC Only
Requirements & Analysis
Design
Coding
Testing

SDLC	ADDIE
Requirements & Analysis	Analysis
Design	Design
Coding	Development
Testing	

SDLC	ADDIE
Requirements & Analysis	Analysis
Design	Design
Coding	Development
Testing	

ADDIE Only
Analysis
Design
Development

SDLC	ADDIE
Implemen-tation	Implemen-tation
Maintenance	Evaluation

Figure 9. Training Development After the Sprint. Adapted from: Norex, Inc., 2016, *Agile Scrum Process.* Copyright 2016 by Norex, Inc.

Conclusion

ADDIE is alive and well, and will probably continue to be so for the foreseeable future. That's because ADDIE is a conceptual framework for instructional design and development, not a model. Many instructional design models have been built from the ADDIE framework. They

include specific steps, tools, and processes for the situations for which they were designed. When the training tools or deliverables change the models change, but ADDIE remains. An example of a model built from the ADDIE Framework is the Successive Approximation Model (SAM), covered earlier, that was designed to develop e-learning courses in an Agile manner.

SDLC is a conceptual framework for systems development, and like ADDIE it too is a fundamental framework on which many models have been built, including Agile Scrum. In both cases, while additional principles and processes are provided via the models that help teams work more effectively in various circumstances, this does not mean that either ADDIE or SDLC are dead, even in an Agile world.

To demonstrate this, two approaches for incorporating ADDIE into the Agile Scrum framework are included. The first incorporates the instructional design work into the sprint itself and it is considered part of the Definition of Done. The second approach has the instructional design work being conducted following the completion of the sprint by another team. Either way, the phases of the ADDIE framework are still completed in the sequential, iterative ways that they always have been.

Good instructional design processes help people create effective training materials, but no tool or model has been created yet that can guarantee it. Instructional design is a skill, like any other, that improves with experience. Qualified instructional designers will continue to use variations of ADDIE and to adapt models, methodologies, and processes including SAM, to help them create effective training. ADDIE lives on!

References

Allen, M. 2012. *Leaving ADDIE for SAM: An Agile Model for Developing the Best Learning Experiences.* Alexandria, VA: ASTD Press.

Bertram, J. 2016. *Agile Learning Design for Beginners: Designing learning at the speed of change.* New Palestine, IN: Bottom-Line Performance.

Bichelmeyer, B. 2005. *The ADDIE Model: A Metaphor for the Lack of Clarity in the Field of IDT.* Retrieved from: http://www.indiana.edu/~idt/shortpapers/documents/IDTf_Bic.pdf

Branson, R. K. 1978. The interservice procedures for instructional systems development. *Educational Technology*, March, 11-14.

Franklin, M. 2006. *Performance Gap Analysis: Human Performance Improvement.* Alexandria, VA: ASTD Press.

Highsmith, J. 2001. *History: The Agile Manifesto*. Retrieved from: agilemanifesto.org.

Hodell, C. 2005. *Basics of Instructional Systems Development.* INFO-LINE, Issue 9706. Alexandria, VA: ASTD Press.

Hodell, C. 2016. *ISD From the Ground Up: A Non-Nonsense Approach to Instructional Design*. 4th ed. Alexandria, VA: ASTD Press.

Kirkpatrick, D. 2016. *The Kirkpatrick Model.* Retrieved from: http://www.kirkpatrickpartners.com/OurPhilosophy/TheKirkpatrickModel.

Molenda, M. 2003. In Search of the Elusive ADDIE Model. *Performance Improvement,* 42(5), 34-36.

Norex, Inc. 2016. *Agile Scrum Process.* Retrieved from: https://www.norex.net/news/thescrumapproachjune2011

Schwaber, K. and Sutherland, S. 2014. *The Scrum Guide ™ - The Definitive Guide to Scrum: The*

Rules of the Game. Westminster, CO: Scrum.org and ScrumInc.

Scrum Alliance. 2016. *Learn About Scrum.* Retrieved from: https://www.scrumalliance.org/why-scrum#why-its-called-scrum