Running head: AGILE INSTRUCTIONAL DESIGN

Agile Methods of Software Engineering should Continue to have an Influence over Instructional Design Methodologies.

Peter Rawsthorne

Cape Breton University & Memorial University of Newfoundland

Partial fulfillment of the requirements for EDU533

Dr. David Lloyd

Saturday, December 10, 2005

Index

Abstract

Since the 1960's computer technologies and related practices and methods have had a significant influence over Instructional Design methods. One of the major trends is the influence of Software Development Life Cycle methodologies over Instructional Design methodologies. This influence is evident in the ADDIE, Dick and Carey, Rapid Prototyping and other Instructional Design methodologies. The discipline of software engineering is going through another methodology paradigm shift; this shift is known as Agile. The Agile methods and practices have much to offer in building upon the existing set of Instructional Design methodologies.

The problem domain

The influence of software engineering methodologies and practices over Instructional Systems Design (ISD) has been ongoing since Computer Based Training (CBT) has been available. During the last forty years new software engineering methodologies have become available. The methodologies that have stayed continue to have an influence over ISD. As defined by Pressman (1997) these methodologies include; Linear Sequential Model (waterfall), Prototyping, Rapid Application Development and the Evolutionary models. All of these models have, with varying degrees of success, worked within ISD. As Douglas (2001) claims, there is also a growing discontent among many practitioners with the ISD methodology. Douglas (2001) continues in saying that traditional ISD has been criticized for being too slow, not necessary, leading to bad solutions and having an outdated world-view. There is also an absence of an alternative to assist in remedying this lack of a strong new design process for ID. Tozman (2004) states that what seemed missing, however, were a discussion about the options or limitations technology provides to the design process. As an answer to these criticisms comes the next iteration of software development methodologies known as the Agile methodologies.

Agile Methodologies

The agile methodologies are pragmatic and focused upon implementing software that meets the customers needs, no more, no less. A lot of what is within the traditional software engineering methodologies is discarded within the Agile methodologies. This new approach is well described within one of the most popular Agile methodologies, Extreme Programming (2001); Architecture is shared and not held by an individual, the customer has direct contact with the programmers not via an analyst, the documentation is minimized and not ritualized, software unit tests are predefined and implemented before software development begins, programmers do not work alone but in pairs, teams are multidisciplinary, software is never considered finished and is always being refactored. It should be considered that the Agile methodologies have grown out of, the need for shortened development cycles; the acknowledgement that software is never finished; that development teams change through time; that technical documentation is rarely read; and that customer requirements change as the project progresses. The philosophy of Agile (2001) is best described by its 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.

Traditional Instructional Design Methodologies

Instructional Design (ID) methodologies have in general been based upon the typical software engineering methodologies. Kennedy (1998) states that these models are representative of common approaches to software engineering…. The models have been selected because the expertise and funding to implement them are generally available in higher education. Two of the most popular of these software engineering based ID methodologies are ADDIE and Dick &

Carey. The linear nature of these two methodologies is best described with a high level review of each followed by a brief summary of how Agile methods could improve these two methodologies. It should be noted that both these methodologies suffer the restraints of their linear roots. As described by Nichani (2002), many reasons are attributed to this prevalent condition—from time and budget constraints to limitations of traditional instructional design. We feel another important reason is the lack of exposure to alternative practices.

ADDIE

As described by Tozman (2004) the term ADDIE is an acronym for Analysis, Design, Development, Implementation, and Evaluation. In ADDIE, the completion of one step is logically fed into the one immediately after it. The ADDIE methodology is very linear and as Kennedy (1998) states, the most problematic in an educational environment. As an improvement upon ADDIE I would suggest the philosophical foundation of the methodology be completely reviewed and replaced by an equivalent Agile type manifesto. From this new foundation new methods and practices would be built to better suit the current restraints within the instructional design environment. Consider time, budget, changing learning theories, increased use of constructivist methods, availability of subject matter experts, changing ID development staff, rapidly changing technology and media channels as the restraints.

Dick & Carey

The Dick and Carey methodology has been very popular due to its continuous improvement with the authors focus on keeping it up to date. With all of the positive effort with this methodology it still “suffers” from its linear roots. The steps of the Dick and Carey methodology are well described by Lee (n.d.);

Stage 1. Instructional Goals

Stage 2. Instructional Analysis

Stage 3. Entry Behaviors and Learner Characteristics

Stage 4. Performance Objectives

Stage 5. Criterion-Referenced Test Items

Stage 6. Instructional Strategy

Stage 7. Instructional Materials

Stage 8. Formative Evaluation

Stage 9. Summative Evaluation

The linear nature of the Dick and Carey methodology is very apparent when the stages are read together. One stage leads to another. Tozman (2004) describes that no one has ever challenged the philosophy that underpins the methodology or disputed the validity of how the methodology supports that philosophy. Agile methodologies are challenging typical software engineering methodologies and therefore they also should be challenging traditional ID methodologies. For Agile to effect positive change upon the Dick and Carey model I offer the following changes;

Stage 1. Curriculum Planning

Stage 2. Identify Learning Themes and Metaphors (Anchors)

Stage 3. Identify Learner Roles

Stage 4. Trawl for Learning Objectives, Modules and Competencies.

Stage 5. Identify Test Cases (Proof of Competencies)

Stage 6 & 7. Pair Programming (Development)

Stage 8. Unit Test (Automated Testing)

Stage 9. Release to Production (Refactor, Refactor, Refactor)

Iterate to Stage 4. within curriculum plan.

Needless to say this offering requires much greater detail and defense. This should be the subject for further research. Take this offering as evidence of the ease and appropriateness that Agile methods can have upon the Dick and Carey methodology.

Software Engineering Influence

A number of significant trends in software engineering have had an influence upon me to believe that Agile is the new methodological foundation required for ID. These initiatives or trends come both from private companies and the public domain. The combination of these trends with Agile create what I consider to be a strong new ID methodology.

Microsoft Solution Framework

The Microsoft Solutions Framework (MSF) exists here for it provides an excellent framework for developing a complete and pragmatic environment to ensure software success. The framework identifies five stages, they include; Envision, Plan, Build, Stabilize and Deploy. This framework also uses a vocabulary which is forward looking and therefore better prepared to address emerging technologies and learning theories. As an example, MSF uses Envision to start, where ADDIE uses Analyze. In a domain that is highly influenced by technology, which ID is, to envision what is needed in the future is a stronger start position than to analyze what has been or is in the present. Having a vocabulary to support a forward looking approach will strengthen the methodology.

Rapid Prototyping

Rapid Prototyping exists as an influence for it introduced the idea of programmers working directly with the end customer (in the case of ID, the learner). This is a very powerful concept which Agile has taken much farther. The roots of Agile are in Rapid Prototyping. By building upon rapid prototyping and the Agile practice of pair programming, teams of multi-disciplinary ID developers with rich media, database, SCORM, Web, and other necessary skills would be brought together with instructional designers to develop the next generation of “courseware”.

Agile

Agile is the new way to build software. It is pragmatic and best described by Fowler (2003);

- Agile methods are adaptive rather than predictive. Engineering methods tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So their nature is to resist change. The agile methods, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.
- Agile methods are people-oriented rather than process-oriented. The goal of engineering methods is to define a process that will work well whoever happens to be using it. Agile methods assert that no process will ever make up the skill of the development team, so the role of a process is to support the development team in their work.

The combination of MSF, Rapid Prototyping and Agile create a new foundation for ID development.

Thoughts on Learning Theory

For an Agile approach to ID to work it must also be aligned with current and emerging learning theory. Starting with Spiro’s (1995) work on Cognitive Flexibility through to the continued deepening of constructivist learning theories, there is a blending of Agile into ID already underway. This blending of constructivism with Agile is directly spoken two by Kennedy (1998), Douglas (2001) and indirectly spoken to by Nichani (2002) and Tozman (2004). Agile methods not only support current and emerging learning theories, they enhance constructivism by involving the learner in the curriculum development process.

Agile Instructional Design

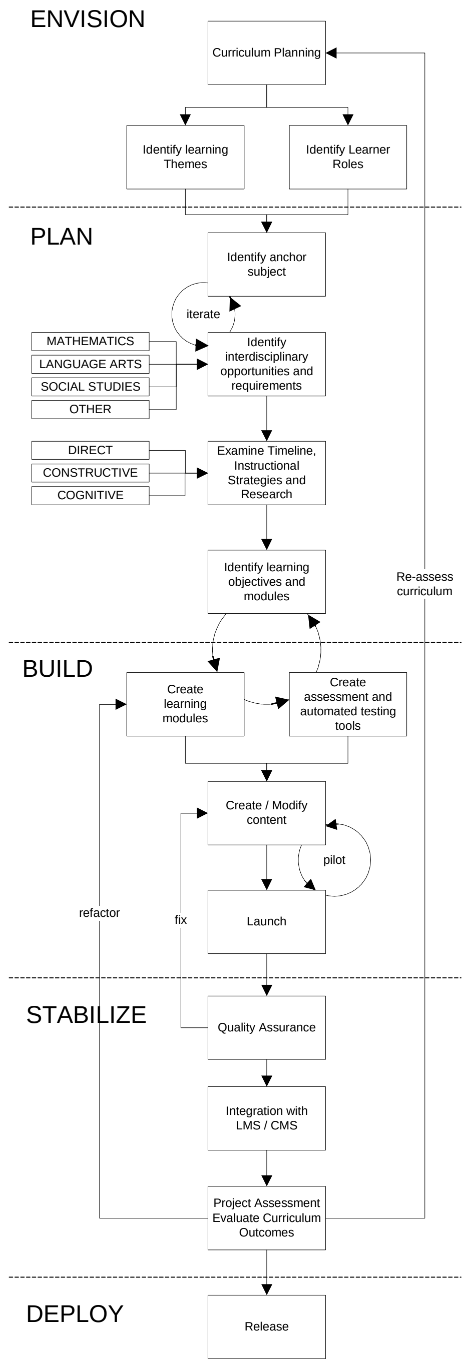


Figure 1: AID Flowchart

The strength of Agile Instructional Design (AID) will come from the early involvement of multidisciplinary teams; the ability of later stage tasks to change previous stage outcomes; the involvement of well defined learner roles; and the ongoing inclusion of emerging technologies and rich media. See Appendix A for a larger sized AID flowchart.

Envision

A curriculum planning meeting is used to envision the current and future curriculum which proposes the overall curriculum modules. Learner roles and context are identified and themes identified to connect learning modules to the curriculum. The curriculum plan is then used to create iteration plans for each individual learning module.

Plan

Planning within Agile Instructional Design is more of a development effort that a planning effort. A multidisciplinary team of software developers, instructional designers, subject experts, rich media designers and learners further define the content of learning modules based upon the identified themes, learner roles, learning theories and interdisciplinary opportunities. Schedules, strategies, an inventory of learning objectives and research requirements are all outcomes of the planning stage.

Build

Building the learning modules is a twofold effort executed by multidisciplinary developer pairs. The twofold effort balances the actual building of the modules with the unit testing of the executable software and the assessment of the learning objectives covered by the module. The pairs combined instructional design and software development skills and knowledge provide a comprehensive ability to build solid instructional modules. As soon as possible in the build process learners are brought in to pilot and evaluate, and potentially improve, the modules.

Stabilize

Stabilization occurs once individual modules are completed and released to Quality Assurance (QA). The process of QA can send modules back to the developer pair for bug fixing. Stabilization also integrates the learning modules with the Course and / or Learning Management Systems. Once stabilization has been complete the modules are deployed to production for learner use. This deployment step also signifies that modules can be refactored, or improved, and also become a part of the curriculum “ecosystem” for use throughout the learning environment.

Deploy

Deployment is the act of putting the learning modules into production for general release. This separation of the deployment activity from all others ensures the quality of the overall release.

The production environment will have its idiosyncrasies which the development team will not be aware. The owners of the production environment (System Administrators) are best suited to manage the deployment.

Conclusion

The Agile software engineering methodologies provide both the philosophical and practical foundation for the next iteration of Instructional Design methodologies. The current set of linear based ID methodologies, though effective, is beginning to show signs of becoming obsolete. The next iteration of ID requires lower costs, faster implementation of new technologies and media sources, less ritual and greater learner involvement. An Agile Instructional Design (AID) methodology would meet the needs of this next iteration along with applying current learning theory more effectively.

References

Agile Alliance (2001). *Manifesto for Agile Software Development*. Retrieved on Nov. 1, 2005 from http://www.agilemanifesto.org/

Extreme Programming. Retrieved on Nov. 20, 2005 from http://www.extremeprogramming.org

Fowler, Martin (2003). *The New Methodology*. Retrieved on Nov. 28, 2005 from http://www.martinfowler.com/articles/newMethodology.html

Gillard, G., Leslie, P. & Rawsthorne P. (2005). *An Instructional Design Methodology to Encourage Student Involvement in Course Design and Implementation*. Retrieved on Nov. 20, 2005 from http://www.rawsthorne.org/bit/medit/ed533/docs/epbsde-v2.pdf

Kennedy, David (1998). *Software Development Teams in Higher Education: An Educators View*. Retrieved on Nov. 13, 2005 from http://www.ascilite.org.au/conferences/wollongong98/asc98-pdf/kennedyd.pdf

Lee, Hee-Sun (n.d.). *Dick and Carey Model*. Retrieved on Nov. 17, 2005 from http://www.umich.edu/%7Eed626/Dick_Carey/dc.html

Microsoft Corporation. *Microsoft Solutions Framework*, (n.d.). Retrieved on Oct. 29, 2005 from http://www.microsoft.com/technet/itsolutions/msf/default.mspx

Nichani, Maish (2002). *Empathic Instructional Design*. Retrieved on Nov. 22, 2005 from http://www.elearningpost.com/features/archives/001003.asp

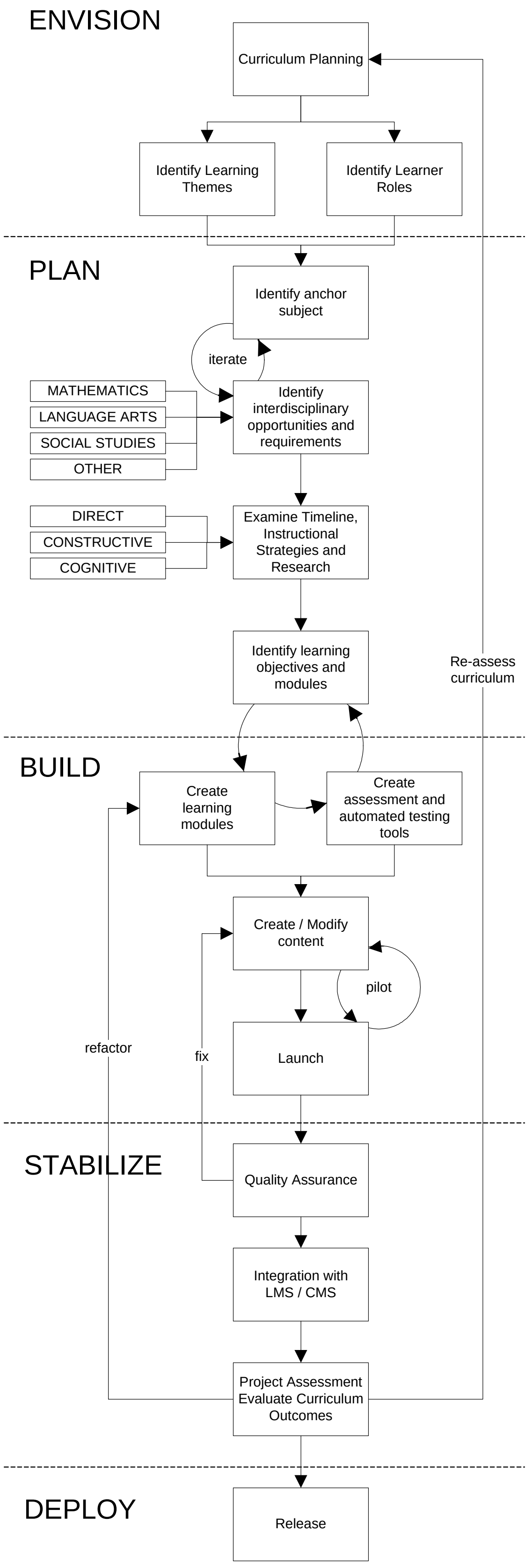
Pressman, Roger (1997). *Software Engineering: A practitioner's approach* (Forth ed.) New York: McGraw-Hill Companies Inc.

Qureshi, Elena (2004). *Instructional Design Models*. (n.d.) Retrieved on November 19, 2005 from http://venus.uwindsor.ca/courses/edfac/morton/instructional_design.htm

Spiro, Rand. (1995). *Cognitive Flexibility, Constructivism, and Hypertext: Random Access Instruction for Advanced Knowledge Acquisition in Ill-Structured Domains*. Retrieved on Nov. 4, 2005 from http://phoenix.sce.fct.unl.pt/simposio/Rand_Spiro.htm

Tozman, Reuben (2004). *Another New Paradigm for Instructional Design*. Retrieved on November 12, 2005 from http://www.learningcircuits.org/2004/nov2004/tozman.htm

Appendix A, AID Flowchart


ENVISION
Curriculum Planning
Identify Learning Themes
Identify Learner Roles
PLAN
Identify anchor subject
iterate
MATHEMATICS
LANGUAGE ARTS
SOCIAL STUDIES
OTHER
Identify interdisciplinary opportunities and requirements
DIRECT
CONSTRUCTIVE
COGNITIVE
Examine Timeline, Instructional Strategies and Research
Identify learning objectives and modules
Re-assess curriculum
BUILD
Create learning modules
Create assessment and automated testing tools
Create / Modify content
pilot
Launch
refactor
fix
STABILIZE
Quality Assurance
Integration with LMS / CMS
Project Assessment Evaluate Curriculum Outcomes
DEPLOY
Release