5-15-2020

A Review and Analysis of Process at the Nexus of Instructional and Software Design

Eric Sembrat

ACCEPTANCE

This dissertation, A REVIEW AND ANALYSIS OF PROCESS AT THE NEXUS OF INSTRUCTIONAL AND SOFTWARE DESIGN, by ERIC SEMBRAT, was prepared under the direction of the candidate's Dissertation Advisory Committee. It is accepted by the committee members in partial fulfillment of the requirements for the degree, Doctor of Philosophy, in the College of Education & Human Development, Georgia State University.

The Dissertation Advisory Committee and the student's Department Chairperson, as representatives of the faculty, certify that this dissertation has met all standards of excellence and scholarship as determined by the faculty.

Brendan Calandra, Ph.D.
Committee Chair

Jonathan Cohen, Ph.D.
Committee Member

Lauren Margulieux, Ph.D.
Committee Member

Maggie Renken, Ph.D.
Committee Member

Date

Brendan Calandra, Ph.D.
Chairperson, Department of Learning Sciences

Paul Alberto, Ph.D.
Dean
College of Education
& Human Development

AUTHOR'S STATEMENT

Eric Sembrat

NOTICE TO BORROWERS

Eric Scott Sembrat

Learning Technologies

College of Education and Human Development

Georgia State University

The director of this dissertation is:

Brendan Calandra

Department of Learning Sciences

College of Education and Human Development

Georgia State University

Atlanta, GA 30303

CURRICULUM VITAE

Eric Scott Sembrat

ADDRESS: 3724 Donaldson Dr NE
Brookhaven, GA 30319

EDUCATION:

Ph.D.	2020	Georgia State University Instructional Technology
Master's Degree	2011	Kennesaw State University Information Systems
Bachelor's Degree	2009	Georgia Institute of Technology Computer Science

PROFESSIONAL EXPERIENCE:

2014-present	Web Manager Georgia Institute of Technology
2012-2014	Web Developer Georgia Institute of Technology
2011-2012	Web Developer Kennesaw State University

PROFESSIONAL SOCIETIES AND ORGANIZATIONS:

2018 - 2020	HighEdWeb
2016 - 2019	WPCampus
2015 - 2019	DrupalCon Conference Committee
2013 - 2014	Graduates in Instructional Technology (GrITS)

A Review and Analysis of Process at the Nexus of Instructional and Software Design

by

Eric Sembrat

Under the Direction of Brendan Calandra, Ph.D.

ABSTRACT

This dissertation includes a literature review and a single case analysis at the nexus of instructional design and technology and software development. The purpose of this study is to explore the depth and breadth of educational software design and development processes, and educational software reuse, with the intent of uncovering barriers to software development, software re-use and software replication in educational contexts. First, a thorough review of the academic literature was conducted on a representative sampling of educational technology studies. An examination of a 15-year time period within four representative journals identified 72 studies that addressed educational software to some extent. An additional sampling of the initial results identified 50 of those studies that discussed software the development process. These were further analyzed for evidence of software re-use and replication. Review results found a lack of reusable and/or replication-focused reports of instructional software development in educational technology journals, but found some reporting of educational technology reuse and replication from articles outside of educational technology. Based on the analysis, possible reasons for this occurrence are discussed. The author then proposes how a model for conducting and presenting instructional software design and development research based on the constructs of design-based research and cultural-historical activity theory might help mitigate this gap. Finally, the author presents a qualitative analysis of the software development process within a large, design-based educational technology project using cultural-historical activity theory (CHAT) as a lens. Using CHAT, the author seeks to uncover contradictions between the working worlds of instructional design and technology and software development with the intent of demonstrating how to mitigate tensions between these systems, and ultimately to increase the likelihood of reusable/replicable educational technologies. Findings reveal myriad tensions and social contradictions centered around the translation of instructional goals and requirements into software design and development tasks. Based on these results, the researcher proposes an educational software development framework called *the iterative and integrative instructional software design framework* that may help alleviate these tensions and thus make educational software design and development more productive, transparent, and replicable.

INDEX WORDS: literature review, case study, critical interpretive synthesis, software development, programming, replication, reuse, cultural historical activity theory, activity theory

A Review and Analysis of Process at the Nexus of Instructional and Software Design

by

Eric Sembrat

A Dissertation

Presented in Partial Fulfillment of Requirements for the

Degree of

Doctor of Philosophy

in

Instructional Technology

in

The Department of Learning Sciences

in

the College of Education and Human Development

Georgia State University

Atlanta, GA

2020

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant 1433280.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

A Review of the Literature on Process at the Nexus of Instructional and Software Design

Similar research and development methodologies can be found in both software design and instructional design. One notable example is the parallel design between the waterfall software development and ADDIE instructional systems design models (Tripp & Bichelmeyer, 1990). This may be because professionals in both fields tend to focus on establishing orderly and replicable processes to help solve large and complex problems (Maher & Ingram, 1989). Perspectives on and utilization of technology within each field, however, can be quite different. While software engineers view a given technology through its technical usability and limitations, instructional designers consider technological affordances as they relate to supporting instructional strategies and goals (Kirschner, 2004). In the past, this perspective has placed the role of educational software in the field of Instructional Design and Technology (IDT) more as a vehicle for the delivery of instructional content, rather than as a research and development goal, or subject worthy of examination in and of itself (Clark, 1983; Oliver, 2011).

Problem

A lack of transparency into the educational software design process may hinder the field from advancing and evolving its educational software design and development practices through the limited ability to replicate successful educational software design and development processes, which in turn complicates the duties and tasks of educational software developers and project managers (Roschelle & DiGiano, 2004). A major premise underlying this review is that reconciling computer science and IDT design perspectives in the existing educational technology literature may provide a path towards consensus, best-practices, progress, and heuristics in educational software development. The rationale for this work is to initiate the process of determining consensus and best-practices at the nexus of instructional and software design.

Software development barriers-to-entry have been identified in educational research. It is an arduous and costly task - software creation is expensive, time-consuming (de Diana & van Schaik, 1993; Sanders, Faesi, & Goodman, 2014), and an ubiquitously team-driven endeavor (Panke, Kohls, & Gaiser, 2007). If a research team does not have software development expertise, searching for assistance with software creation can be challenging. Subject-matter-experts frequently turn to institutional resources or the technical expertise of a colleague or student, which may not be sufficient for quality product development (Ormel, Pareja Roblin, McKenney, Voogt, & Pieters, 2012). The loss of technical expertise or resource presents additional risk, as replacement requires sufficient documentation, on-boarding, and prior skills to pick up the project (Jackson & Brannon, 2018). Lacking availability to external resources or expertise, identifying and selecting existing software through search engines, code repositories, or published articles that match the instructional goals of a complex project can be an arduous task (Hucka & Graham, 2018). Identification of consensus between instructional and software design could help minimize the significant resources required to create educational software from scratch and without guidance.

Purpose

This review was conducted in order to explore the breadth and depth of educational software design and development processes, and educational software reuse reported in the IDT literature. The intent of this review is to uncover possible gaps in software development reporting within the field and to propose solutions for future endeavors that can more accurately report software development from a reuse perspective.

Relevant Literature

Operational definitions

The term ‘design’ in IDT and computer science are centered in different contexts. For this reason, the authors will use the terms ‘instructional design’ and ‘software design’ in order to reduce ambiguity (Oncu & Cakir, 2011). Software design for our purposes, refers to the relationship between a software problem or goal, a developers’ own prior experiences, and the strategy to solve the problem or goal – the blueprints and resource planning (Gaydos, 2015). Software development, alternatively, refers to the nuts-and-bolts of software creation: data structures, algorithms, and computer code (Zhu, 2005) – the consumption of blueprints and resources to create. For this literature review, software design does not explicitly refer to (but can include) any visual or user-interface design considerations. Instructional design refers to the activation and support of learning through a set of events constructed to facilitate learning (Gagné, Briggs, & Wager, 1991) by translating learning principles into instructional materials, activities, and evaluations (Smith & Ragan, 2005). Reporting and discussion of instructional content is outside the scope of this literature review.

Critical interpretive synthesis

Critical interpretive synthesis (CIS) is a literature review method developing synthesized results from a wide range of research evidence, both qualitative and quantitative, through an interpretive process (Flemming, 2010). Sourced from meta-ethnology and techniques from grounded theory (Barnett-Page & Thomas, 2009), literature review in CS pairs induction and interpretation processes in the development of concepts and associated theories (Dixon-Woods et al., 2006), resulting in an interpretive model of a phenomenon sourced from evidence in existing literature (Wolgemuth, Hicks, & Agosto, 2017). Search criteria in CIS focuses on the identification and selection of a diverse set of sampled articles, intended to represent the variation found in literature (Morrison, Yardley, Powell, & Michie, 2012) without an exhaustive summary of all literature, primed for practicality and time availability for unmanageable literature search scopes (Dixon-Woods et al., 2006).

CIS leverages the concept of synthetic constructs to compare and analyze individual studies' reporting to the body of literature. Synthetic constructs define the transformations of evidence within individual studies into more-conceptual forms applicable for generalized comparisons, allowing for broader, higher-level interpretations while also pulling in disparate aspects from individual studies (Barnett-Page & Thomas, 2009). Inherited from meta-ethnography, research questions in CIS are iteratively developed, allowing opportunities for refinement and renegotiation during the review process (Flemming, 2010) through the application the lensed analysis of synthesized constructs to the analyzed literature. The output of CIS results in a synthesizing argument – a generalized, formalized conceptual framework constructed to help understand a phenomenon, sourced from synthesized constructs' interpretations from all reviewed evidence.

Method

A thorough literature review guided by CIS was conducted to examine the reporting of software design and development in instructional design and technology research. Data was collected from peer-reviewed academic journals through the application of the systematic literature review model (SLRM) synthesized by Kangas, Koskinen, and Krokfors (2017) from prior work. The SLRM is composed of four phases bounding the literature review: defining research questions, bounding inclusion and exclusion criteria, selecting databases, and defining search terms. A visualization of the literature review process is available in Appendix and formatted per the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) flowchart visualization to help provide additional transparency and clarity to the overall review strategy and an accessible snapshot of the scope, processes, and findings (Liberati et al., 2009). The format of the PRISMA flowchart follows the work of Mertz, Kahrass, and Strech (2016).

Research questions

The following research questions guided the data collection of the literature review:

- RQ1. How is software design and development detailed in IDT literature?
- RQ2. How is software re-use portrayed within IDT literature?

Inclusion & exclusion criteria

To examine the utilization and analysis of software development in IDT research, papers published from 2003-2018 in Computers & Education, Educational Researcher, Educational Technology Research and Development (ETR&D), and International Journals of Designs for

Learning were analyzed. All journal articles from these sources were considered for inclusion. The 15 year selection of 2003 to 2018 corresponded with the establishment and normalization of online interaction and collaborative learning (Zawacki-Richter & Naidu, 2016). Four specific journals were chosen to provide a diverse sampling of papers to represent the broad variation found within literature (Morrison et al., 2012). Extending the literature identification process of Margulieux, Ketenci, and Decker (2019), specific journals were selected to represent IDT due to their scientific rigor (articles were peer-reviewed, and peers ranked the journals as top-tier rigorous journals) and to represent separate facets of research foci in its selected journal articles. Table 1 outlines the submission criteria, explicit exclusions, and scope for each journal.

Table 1

IDT Journal Scope & Perspectives

Journal title	Scope	Submission criteria	Exclusions
Computers & Education	"[Increasing] knowledge and understanding of ways in which digital technology can enhance education".	Context of use, user/system interface, usability issues, user experience evaluations, and impacts on / implications for teaching and learning.	Detailed information on implementation architecture.
Educational Researcher	"General significance to the education research community and that come from a wide range of areas of education research and related disciplines".	"Major programmatic research and new findings of broad importance widely accessible", in the areas of feature articles, reviews, essays, briefs, and technical comments.	n/a
ETR&D	"General significance to the education research community and that come from a wide range of areas of education research and related disciplines".	Development: "planning, implementation, evaluation and management of […] instructional technologies and learning environments". Research: application of technology or instructional design in educational settings. Applied theory, analytical, practical research topics. Cultural and Regional Perspectives: enhance learning, instruction, and performance	n/a
International Journal of Designs for Learning	"Design cases documenting interventions (artifacts, environments or experiences) created to promote learning".	"Critical and/or interesting decisions made during the design process and their results in the intervention, key aspects of the design process as they are relevant to the form of the intervention, and transparent discussion of problems and/or failure analysis relevant to the intervention and its design".	n/a

Note. Content adopted from journal submission webpages (Computers & Education; Educational Technology Research and Development; International Journal of Designs for Learning; Researcher).

Both Educational Researcher and ETR&D provided a scope of general significance to the educational community from wide areas of disciplines and research, serving as broader-lensed journals. The submission criteria outlined in ETR&D, unlike the broader criterion found in Educational Researcher, focused on areas prime for software development studies. The Development track focused on the creation of instructional technologies and learning environments, whereas the Research track focused on the application of developed instructional technology or design into practice. Submission criteria found for Computers & Education provided an intersection between the fields of educational research and computer science, focused on user and system interface designs, contextual use considerations, and usability issues, but explicitly excluding implementation architecture. Submission criteria found for International Journal of Designs for Learning mirrored software development processes with the addition of design detail and intervention documentation to promote learning. The inclusion criterion and scope of each journal suggested that software development-focused educational research could be published.

Additional journals were not chosen due to two criterion. First, database searches on a wide swath of educational journals on a topic area or search term rife with ambiguously-scoped terminology (“software” and “development”) would generate voluminous records unmanageable to the capacity of the author (Dixon-Woods et al., 2006). Identification of the operational definitions underpinning this research revealed software development-focused terminology ambiguity and variation, which could not be accurately assessed in a voluminous record set. Secondly, the goal of the review was to uncover how educational software design and development processes were detailed in IDT literature, rather than within education and education-adjacent fields. This consideration bounded careful journal selection adhering to high-quality article review and selection criterion and journal reputation within IDT.

Databases & Search Terms

Database searches and archival retrievals were conducted for each journal for review and synthesis. For the *International Journal of Designs for Learning*, a search was conducted within the Directory of Open Access Journals database with the following search criteria, returning 5 results:

- Search query: “software development” in all text
- Content Type: Research Articles
- Publication Date: January 2010 (corresponding to the first edition) to March 2018

For Computers & Education, a database search was conducted within the ScienceDirect database with the following search criteria, returning 132 results:

- Search query: “software development” in all text
- Content Type: Research Articles
- Publication Date: January 2003 to March 2018

For Educational Researcher and ETR&D, a database search was conducted within the JSTOR database with the following search criteria, returning 30 results:

- Search query: “software development” in all text
- Content Type: Journals
- Publication Date: January 2003 to December 2014 (the latest date indexed from JSTOR database)

The searchable JSTOR database excluded articles in a three-year period from search date for Educational Researcher and ETR&D, leaving a data gap in applicable articles from January 2015 to March 2018. A supplemental search was conducted through the SAGE Journals archive to obtain journal articles between the exclusion date (January 2015) and the search date (March 2018). Due to SAGE Journals archives’ inability to search through journal articles, all journal articles were manually exported. This export resulted in an additional 183 (Educational Researcher) and 253 (ETR&D) articles. A total of 603 articles were obtained from the summation of all retrieval methods.

General Inquiry (Phase 1)

Each of the 603 articles was analyzed using the search, retrieval, and validation process adapted from Barroso et al. (2003):

1. Read the title, abstract, and metadata for inclusion coding:
    a. Does this article deal with the development and usage of software?
        i. If not, exclude article
2. Read full article for inclusion in coding book following the research questions.
    a. To satisfy RQ1, manually code how the software development process was re-ported
        i. **Reported development process**: study categorizations as reporting on (software) development, (software) design, and/or project team process.
        ii. **Development team**: self-reported (explicitly or implicitly) the software developer.
            1. **Developer Category**: categorize identification of developers into internal (within the research team), external (consultant, staff member), or undisclosed.
        iii. **Evidence of software reuse**, satisfying RQ2: summary if the study dis-cusses re-use in the development, iterative, or future goals of the software.
            1. **Explicit Reuse**: summary if the study explicitly reuses software in the design and development of the software in the study.
        iv. Identify specifically-stated **research scope exclusions** or boundaries: summary of any areas that the authors explicitly ignored for the study, due to additional reporting or paper scope.

b. Describe a short **description of end-result software**: a high-level description (name, goal) of the software

    i. **Development language**: programming languages or tools leveraged in development of the tool, dissected into three additional aspects:

    ii. **Platform**: classifications of the software platform for tablets, personal computers, mobile devices, web browsers, video game consoles, or AR/VR.

    iii. **Custom developed**: if the software was custom built rather than repurposed from existing software.

    iv. **Research methodology**: overall research methodology guiding the study.

The article was then collated into a coding book. As described by Randolph (2009), a coding book collects recorded data for each article, including supplemental data that may influence research. Recorded data includes the presence of CIS' authorial voice – evidence not visible or auditable, but interpreted through the author's lens, background, and expertise (Dixon-Woods et al., 2006) – present within synthetic constructs and the mapping of software design concepts to instructional design studies. Coded entries were appended with selected text quotes and summarizing notes to validate coded interpretations and build trustworthiness (Elo et al., 2014).

Seventy-two articles met inclusion criteria. These studies encompassed the following research methodologies: design-based research, experimental design, case study, usability study, co-design, evaluation study, naturalistic design, formative evaluation, and mixed-methods design. The educational software products in these studies spanned a wide variety of use-cases, including medical simulations, tutor agents, instructional visualizations, integrated learning environments, and role-playing games. The full coding book is available as supplemental material.

Additional Inquiry (Phase 2)

A large number of studies in the initial analysis phase excluded self-reporting of programming language details (40), software developers (32), and the software design, development, and process (35) reporting, as compared to the high number of self-reported custom software development (68) studies. This gap in detailing the creation of custom developed software revealed a dichotomy – studies overwhelmingly leveraged custom software in the creation of interventions, yet the traditional software creation inquiries (questions such as ‘who developed?’, ‘what programming language or toolkit?’, or ‘how was the development managed?’) were not adequately detailed. This exclusion resulted in custom software interventions situated in IDT research which could not be re-created. Additionally, this gap suggested further investigation on studies which included either evidence of custom software development or software re-use that also self-reported software design, development, and process. As a consequence, the author applied an additional iteration of review to further understand custom development or re-use software reporting in IDT research.

A need for a strengthened definition of software re-use emerged throughout the preliminary review phase. The re-use criterion, “discuss[ing] re-use in the development, iterative, or future goals of the software”, did not properly bound what software re-use could entail. A framework identified in the preliminary review phase provided an opportunity to refine the concept of re-use. Cordero et al. (2015) leveraged the Bannan-Ritland (2003) integrative learning design (ILD) framework as a lens for IDT software development. The ILD framework is a concatenation of product design, usage-centered design, educational research, and innovation development, and instructional systems design processes (p.22). The consolidated ILD framework is shown in Figure 1.

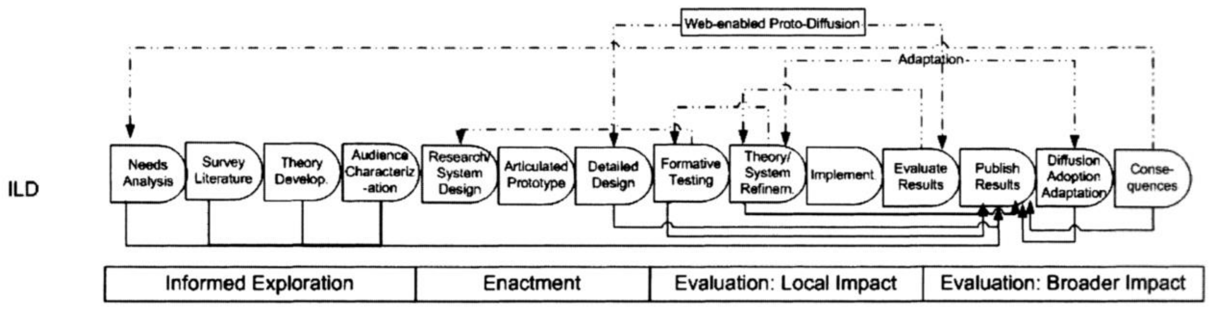


Figure 1. Bannan-Ritland's (2003) Integrative Learning Design framework.

The ILD framework separated software re-use into three categories: diffusion, adoption, and adaptation. Diffusion referred to dissemination of software into a wider audience via the web. ILD envisioned diffusion as a method to solicit supplemental user feedback, resulting in additional design, development, and testing by the original developers (p. 23). The open-source nature of software distribution allows for the software development process to be extended, through source-code publishing, to any interested person to continue, expand, or reiterate upon. Adoption referred to the full re-use of the software without any informed exploration – a translation of a custom-developed software to replication studies. Like adoption, adaptation referred to the re-use of the software, but with secondary studies exploring a variated re-use.

As RQ2 originally identified educational software re-use in broad terms, ILD's definition of re-use as three separate levels (diffusion, adoption, and adaptation) facilitated the addition of supplemental questions extending the original research questions:

- RQ1. How is software design and development detailed in IDT literature?
- RQ2. How is software re-use portrayed in IDT literature?
    - RQ2.1. How is software diffusion portrayed in IDT literature?
    - RQ2.2. How is software adoption portrayed in IDT literature?
    - RQ2.3. How is software adaptation portrayed in IDT literature?

The 72 studies of the first phase were reduced to those which reported software development or reported software reuse, which excluded 22 studies. Studies were excluded if no software reporting was found in the study (software design, development, or process were all excluded) and no evidence software re-use was found. These exclusions resulted in 50 studies, which were re-coded with additional data to answer the RQ2 sub-questions:

1. **Diffusion**: identification of a (digital) location where the software is available to download or access.
   - **a. Diffusion as shared ownership**: evidence of collaborative, shared development resulting in shared ownership, open-source software.
   - **b. Diffusion as distribution**: the traditional definition of diffusion as defined by ILD.
   - **c. Post-development abandonment**: self-reported evidence of post-development deprecation of software code.
2. **Adoption**: identification of rote software re-use within the study.
   - **a. Full software adoption**: the traditional definition of adoption as defined by ILD.
   - **b. Additional software adopted during design and development**: adoption of a software module, or additional software.
   - **c. Sourced adopted software**: categorization of adopted software as originating from research, a software application or plugin.
   - **d. Post-development adoption**: self-reported evidence of post-development adoption by additional research.
3. **Adaptation**: identification of variated re-use of existing software within the study.

a. **Sourced adapted software**: categorization of adapted software as inspiration, module, or software application.
b. **Post development adaptation**: self-reported evidence of post-development adaptation by additional research.

Review of the initial phase of analysis revealed unclear differentiation between adoption and adaptation when scoped to software modules. The concept of variated re-use is a foundational component of software modules; their incorporation into a larger product can often be variated. Software modules can require additional programming code for enabling basic functionality, while others are intended to be used stand-alone and provide data output to a consuming custom-developed software application. To clarify this differentiation, the author defined a synthetic construct for software categorization as an application (standalone software) or module (a piece of software requiring additional customization to function).

Revised synthetic constructs for adoption and adaptation were provided for software applications and modules. For software applications, adoption was scoped to research which did not directly alter the software through additional custom programming code. Adaptation was scoped to research which extended existing software beyond its original capabilities, requiring additionally-written software or hardware. Adaptation was additionally scoped to software utilized as inspiration – adaptation through variated replication of its interface or design. This scenario is similar to the video game concept of a ‘spiritual successor’, where a new video game may replicate components or best-practices of an existing game into a new, un-connected game. For software modules, adoption was scoped to research which did not require additional custom code to extend the module functionalities. For example, usage of a third-party software AR/VR toolkit would be considered adoption if no additional programming code was required to extend the toolkit beyond its current usage. Adaptation was scoped to research which required additional custom code to extend functionality to a new domain or use-case.

Results

General Inquiry (Phase 1)

Hardware platforms.

All 72 studies were analyzed for the self-reported hardware platforms of the developed software. An overview of the platforms is described in Table 2. Each category was identified and defined as the platform appeared within the review.

Table 2

Self-Reported Hardware Platforms

Hardware Platform	Studies found
Tablet	6
Video Game Console	1
Web Application	32
Mobile Application	16
Desktop / Laptop Application	23
Augmented / Virtual Reality	8
Total	84
Blank / None	0

IDT research reached a large population of devices – from device-agnostic (web) to device-dependent (AR/VR, video game console), and all the operating system ecosystems in between (mobile, tablet, desktop, laptop). The total platform count included software that existed on more than one hardware platform. Multiple studies included software that utilized augmented or virtual reality (AR/VR) in order to provide instruction – AV/VR as the visual apparatus paired to a mobile device or computer. Multiple studies also leveraged both a medium of instruction (web) and a physical tool such as a tablet or mobile device. The wide range of hardware platforms illustrated the ubiquity of software platform reporting in educational research, but simultaneously highlighted low software replication or re-use value. Hardware platform reporting did not provide meaningful software creation process detail except that it was designed and expected to work on a specified hardware device. Hardware selection and identification did not necessarily entail any judgements, insights, or detail into the software development process. There is no implication or evidence suggesting any parallelism between software development processes of two projects sharing similar hardware platforms. Instead, hardware platform reporting defined and articulated the methods of user-interaction, providing opportunities for a high-level description of how end-users interact with the software. Uncovering software reporting primed for re-use required a deeper dive into the software creation process itself.

Software programming languages and tools.

All 72 studies were analyzed for the self-reported programming language. The results of this analysis is described in Table 3.

Table 3

Self-Reported Software Programming Language or Platform

Software language	Studies Found
Software Programming	
HTML5	2
Ruby on Rails	1
PHP	1
Objective C / iPhone SDK / related libraries	1
JavaScript	1
Unity3D / Unity	4
C++	1
Java / Android SDK / related libraries	6
Flash	1
ColdFusion	1
C# / .NET	4

Perl	1
Total (Software programming)	24
Database Programming	
mySQL	3
Microsoft SQL Server	1
Total (Database programming)	4
Visual Programming Application	
Microsoft Agent Environment	1
Vuforia & ManySense	1
NetLogo & Flash	1
SecondLife	1
EmuCV	1
SimQuest, Loquendo, Elcklyc	1
Neverwinter Nights 2 mod	1
eAdventure software framework	1
Macromedia Director	1
Total (Programming Tools)	9
Undefined Programming	
“a series of API calls”	1
Total Programming Tools & Languages	38
Total Studies	32
Blank / None	40

Programming language self-reporting within the sample fell into two distinct camps of identification and absence. Programming tools consisted of pre-existing full-featured software with editor tools (such as SecondLife and Neverwinter Nights) or an assortment of authorware tools designed to scaffold software development by providing a foundation software framework and editor user interface (Boyd, 1993). While evidence of a few authorware tools were identified in the sample, a majority of studies detailed programming language usage. As one study noted, programming language usage reduced development costs and provided an opportunity for content experts to be closer-situated to the development environment (Borro-Escribano, Del Blanco, Torrente, Alpuente, & Fernández-Manjón, 2014). The breadth of programming languages exemplified the variation within IDT research. Reported programming language usage spanned from game development, mobile operating systems-specific, deprecated or abandoned tools, web-based products, general purpose, and low-level code. Few studies leveraged database storage and querying, an aspect vital for web-based platforms. No singular programming language or visual programming application was found to be ubiquitous or preferred across studies. As an example, game development studies centered around Unity, a library and toolkit providing programming language shortcuts primed for 3D video game development, whereas web development studies spanned across a range of programming languages and tools.

The wide range of tools and programming languages provided evidence of a fractured and segmented development environment, as it severely limited value in reporting code samples and snippets. Code syntax, libraries, and structure could not be easily translated between visual programming languages, traditional code-oriented programming languages, and database programming. Code snippets would have limited value to researchers primed to replicate or reproduce the software – the effort would be akin to an author rewriting a novel based on quoted text or a paragraph of plot summary. It would be infeasible to expect IDT research standardize on one platform, toolset, or language, as programming languages were designed to provide shortcuts for particular types and variations of code. Gaudel (1981) and Leupers (2002) detailed the foundational knowledge on the decomposition process of programming languages into shared computer code. As a summary, standardization to a single programming toolset or language for the field would, both for software developers and instructional experts, vastly increase the resources and time required in software creation, requiring a massive amount of additional work added to an already resource-intensive process. For example, a programming language primed for 3D visualizations (such as Unity) provided unique libraries and shared code components built for that primary purpose. Rewriting those libraries, for example, within a web-oriented programming language would be unrealistic and wasteful.

While programming language reporting did not provide any real detail of a software development process, it highlighted the increasing pace of standards and deprecation in software development. Multiple studies worked with historically relevant programming languages (Flash, ColdFusion, Perl, Ruby on Rails) that had fallen out-of-favor for more modern-day mobile-oriented programming languages (Android SDK, Java, Objective C). Even programming tools themselves (Macromedia Director, Neverwinter Nights 2, Microsoft Agent Environment) dated themselves into obsolescence. Comparatively, while base-level programming languages (HTML, PHP, C++, JavaScript), were historically relevant, those components also served as foundations for modern programming language framework such as Node, Symfony, React, and VueJS. Consistency in reporting on programming languages could provide best-practices on effective programming languages for particular types of educational software development. However, that self-reporting would provide only traces of software development. To understand more about reporting software development, it was valuable to examine the extent of custom software development comparative to existing software analysis.

Software customization or reuse.

Studies were analyzed for the self-reported re-use or custom development of software in each study. An overview of the customization is described in Table 4.

Table 4

Self-Reported Software Creation Method

Software re-use	Studies found
Reuse of Existing Code	7
Custom Development	68
Total	75
Blank / None	0

There was no reliable pattern or standard definition of custom software development. To assess presence of custom development, a study met the minimum requirement for categorization if software development descriptions contained any permutation of "researchers developed the [environment]", as seen in studies such as Huang and Huang (2015) and Paek, Hoffman, and Black (2016). Out of the seven studies re-using existing code, only three studies used a combination of pre-existing software and custom plugin development. These studies identified two combination scenarios:

- Existing software that provided a native authoring tool (Chen, Wong, & Wang, 2014)
- Existing software that provided programming-level hooks or work-arounds (H.-M. Sun & Cheng, 2009; van der Schaaf et al., 2017)

The wide presence of custom development over explicit re-use highlighted a foundational struggle in the field – the plight of replication studies in academic research. This plight is mired by multiple factors: fears of recreating ‘exact’ replication (Loui, 2015), limited extensions for existing research (Plonsky, 2015), an over-emphasis of ‘gadgeteering’ over practical theory and practice (Finn, 1953), and a combination of institutional, editorial, curricular, and personal perspectives (Porte, 2013). These fears stoked an environment primed for custom developed software, facilitating usage of novel, new ‘gadgeteered’ solutions rather than extending or confirming already-created educational software. This perspective stymied the 3 R’s defined by Gage and Stevens (2018), rigor, replication, and reproducibility, to increase research relevancy in scientific domains and society. Additional analysis was needed to evaluate the context of custom developed educational software. It was possible that custom software afforded more opportunities for discussion, detail, and analysis of the development process, affording more generalizable data on the software creation process; this guided the next aspect of inquiry.

Software design, development, and process.

All 72 studies were analyzed for reporting of software design, development, and process within each study. The results are described in Table 5.

Table 5

Self-Reported Software Design, Development, or Process

Software development self-reporting	Studies found
Software Design	28
Software Development	16
Software Team and Process	13
Total Studies Reporting	37
Total Studies Not Reporting	35

A synthetic construct was established to gauge the reporting of the software design, development, and team process based on the study's creative process (Dixon-Woods et al., 2006). The construct was guided by the amalgamation of the following aspects:

- Construct of 'software design': visual design considerations.
    - Included any discussion of user interface, screenshots, high-level software design models, and user experience detail, as these aspects detailed how the programmed software was presented to its target audience.
    - Excluded any high-level overview of software features or goals.
- Construct of 'software development': programming-related coding.
    - Included any discussion and process regarding information architectures, code samples, low-level system modeling, and UML diagrams - as these aspects detailed the programming perspective and strategy.
- Construct of 'software team and process':
    - Included any discussion of the design and development timeline or iterative development process.
- Construct of 'No reporting': the study presented the software as if it was spontaneously acquired or developed. Tergan (2006) described this perspective as a singular focus on instructional-focused analysis, forgoing any considerations to the development of the software.

The terms software design and development were not explicitly scoped or defined within any individual article. This limitation revealed that ill-defined and ambiguous definitions may hinder software reporting aspects. Visual software design was widely utilized as a meaningful perspective into the inner-workings of the software development process. A majority of articles which reported an aspect of the software development process showcased software design as a reflection of the software's creation. This was not too surprising, as there are fields of research such as user experience, usability, and accessibility, which focused on design language and their translations to end-users. These fields of research examined how end-users interpret and use the software, and examined barriers or scaffolds to locate and consume instructional design. Similarly, because design is primarily visually-attuned, it was easily-translatable to a journal article or paper. Screenshots, design documents, and interface mockups served as "gadgeteered" results, highlighting aesthetic appeal over underlying functionality. Prior to the first iteration of analysis, there were considerations for software design to exclude visual design. However, software design was defined as primarily focusing on visual aspects like user interfaces, rather than data structural design or project goal design. An effort to consolidate and frame these concepts from an educational perspective would be incredibly valuable, if only to properly guide analysis and reporting for re-use across IDT using a standardized vocabulary.

A large number of studies explicitly reported that software development fell outside the scope of this review, deferring to either external documents (Winters & Mor, 2008) or prior research (Depradine & Gay, 2004; Hirumi et al., 2016; Khan, 2008; Leenaars, van Joolingen, Gijlers, & Bollen, 2014; Liu, 2013; S. W. Park & Kim, 2014, 2016; Verdú et al., 2017; Waight, Liu, & Gregorius, 2015; Winters & Mor, 2008). The prior sections of analysis highlighted the small presence of software development reporting: code samples and information architectures would not be able to describe the software development process – they only described aspects of the resulting software output. Moreover, Computers & Education explicitly excluded journals highlighting information architecture-like analyses in its submission criterion, as seen in Table 1. This explicit exclusion implied two considerations: information architectures had very narrow value to the larger field of research, or/and its usefulness fell outside of the scope of educational research. Code visualizations like UML diagrams and information architectures would be comparable to a table of contents. Code descriptions were heavily generalized but broadly scoped across the project. From the perspective of re-use, information architectures provided little value beyond a high-level review of programmable components. Additionally, there was evidence that educational research did not prioritize software efficiency. Developed software was found to not be stable (Laine, Nygren, Dirin, & Suk, 2016) or flexible for real-world use (Modell, 2014), or too early in development due to ineffective choices in programming languages (Laine et al., 2016), limited project scope (Dagdilelis, Evangelidis, Satratzemi, Efopoulos, & Zagouras, 2003), or just preliminary findings (de Jong et al., 2012; Wei, Weng, Liu, & Wang, 2015). These quality-control issues forced reconsiderations of the value of full software development sharing in cases of inefficient or buggy code. Stepping away from code examination, it was crucial to examine who was developing the software to better understand how software development was situated alongside IDT expertise.

Software developers.

All 72 studies were analyzed for self-assessment of who developed the software. The results are described in Table 6.

Table 6

Self-Reported Software Developer(s)

Self-Reported Developer Role	Studies Found
Internal Developers	30
External Developers	10
Total Studies Disclosing Developers	40
Total Studies Reporting an Undisclosed Developers	32

Data coding was bound to set criteria, noting ambiguity in identifying a software developer as internal or external to the research project. Studies were coded as reporting internal developers only if the study explicitly self-reported the authors or a member of the research team as the developers. Beyond explicit declarations of external software developers, references to ambiguously-scoped software programmers were categorized as external developers, as their relationship between the researchers could not be determined. Studies failing to include a reference to who developed the software were coded as containing undisclosed developers. Reporting of software developers was closely divided between those that disclosed and those that did not. However, multiple studies cited software developers solely as part of the supplemental acknowledgement, which provided very little context and knowledge of how developers interfaced with the project team members. Exclusion of software developers within research about the software deprived similar studies (defined by either scope, scale, research topic, or instructional goals) valuable data of the researcher/developer relationships and knowledge negotiations. Detail and analysis into software developers would provide value to cite or situate knowledge and expertise from a complementary field (like computer science) into the study. More importantly, the placement of software developers within the study writing process itself would allow their voice (and by association, their knowledge and expertise from that complementary field) to sit alongside educational-research, providing additional evidence and validity to the software creation process. Framing analysis from this perspective, however, may prove to be a challenge within IDT. Referencing software development knowledge from textbook publications to kick-start development highlighted the difficulty.

The extent of reporting for studies leveraging external developers varied extensively. External developers included colleagues across multiple fields within an institution, paid development roles, computer science experts, an international project team of programmers, individual members with undefined relationships to the research project, and generically-defined external programmers. As described earlier, the software development process was a non-trivial process that was team-work driven, time-consuming, and expensive. Given the resources required, the variation in external development resources was surprising. If external developers were solely scoped to software agencies and other ‘think-tank’ subject-matter-expert vendors, an argument could be made about external developers bringing in a vast swath of expertise given a cost value (either as resource, time, or monetary). However, external developers encompassed a larger field of academic expertise, international scaled experts, and institutional colleagues. This highlighted a possible gap that the cost for software agencies may be a barrier-of-entry as compared to freelance-scoped developers. The knowledge in order to validate this possibility, however, would be impossible to obtain without additional detail on the software development processes in each study.

Re-use Inquiry (Phase 2)

Further analysis was needed to understand how studies leveraged software adoption, adaptation, and diffusion. Synthetic constructs of adoption, adaptation, and diffusion were established, grounded in ILD's structural steps, defined by Bannan-Ritland (2003), to bound identifications within each study:

1. **Diffusion**: evidence of diffusion only if the author(s) provided a digital presence for the project and the developed software.
2. **Adoption**: evidence of adoption in two scenarios: (1) leveraging existing software in the creation of the software, or (2) explicitly-defined evidence of post-development adoption of the developed software.
3. **Adaptation**: evidence of adaptation in three scenarios: (1) repurposing components, content, design from other software, products, or studies in a new scenario, (2) explicitly-defined evidence of during-development adaptation of software, or (3) explicitly-defined evidence of post-development adaptation of the developed software.

ILD's definitions of adoption and adaptation were adjusted to scope its presence to encompass both within a study (during-development) and outside the original study (post-development), with the premise that a study could either/both frame its software development as a consumer or producer. Studies framed from software consumption allowed for ILD's original definitions of adoption and adaptations, where the study acted upon previously-created software in its own creation. On the other end, studies framed from software production allowed for the emergence of adoption and adaptation evidence in post-study discussions. In addition, software development analysis could be framed both production and consumption. Software could simultaneously be modular and componentized, and adopt many small software packages in the creation of unique, larger-scaled software. Individual software components, or modules, are commonly shared (disseminated), repurposed (adaptation), and applied (adoption) within software communities such as open-source software. Understanding the nesting doll-like capacity of software development, where a new product could be produced from the adoption and adaptation of existing software, was crucial to properly bound the synthetic constructs.

Software re-use components.

50 studies were analyzed for presence of software re-use as defined by the above synthetic constructs. The results are described in Table 7.

Table 7

Software Re-use as Defined by the Integrative Learning Design Framework (n=50)

Software re-use	Studies found
Adoption	18
Adaptation	15
Diffusion	9
Adoption and Adaptation	3
Diffusion and Adoption	2
Diffusion and Adaptation	0
Adoption, Adaptation, & Diffusion	4
Total studies reporting	29
- Reporting only one aspect	20
- Reporting only two aspects	5
- Reporting all three aspects	4
No reporting	21

The sampled articles were grouped into studies reporting re-use and not. Coupled with the instances within the sample of re-use aspect combinations, these findings provided insight into the difficulties for IDT software to persist and evolve within a learning ecology – configuration sets of activities, resources, and relationships situated within co-located spaces which provided learning opportunities (Barron, 2004). While the second phase of articles inclusion criterion were bounded to software development reporting, this schism of re-use and no reporting provided evidence that software development-focused research does not maximize re-use. The fewer instances of software diffusion, compared to adoption and adaptation, highlighted the problem of locating similarly-developed software as described by Hucka and Graham (2018). Their research focused on locating applicable software for scientific research, finding a large presence of "look[ing] in the scientific literature to find what authors use in similar contexts" alongside web searches and colleague advice. Diffusion was defined to situate within both scientific literature and web searches, in that the process placed a breadcrumb trail from the study to the applicable website or repository.

Similar findings of adoption and adaptation evidence were found within the sample size, mirroring the structure of academic research. Academic research served as an appendix, extension, and expansion of the existing learning ecology (Cobb, Confrey, diSessa, Lehrer, & Schauble, 2003). It was not surprising that adoption and adaptation were the most prevalent aspects of re-use, as these software projects were the digital products of a growing learning ecology within IDT. However, additional analysis was necessary to understand and situate the presence of adoption and adaptation, as the synthetic constructs provided multi-faceted bounds by which to collect applicable data across the software development lifecycle.

Software diffusion.

50 studies were analyzed for presence of software diffusion. Sub-types emerged from analysis to categorize and group similar diffusion types. The results are described in Table 8.

Table 8

Self-Reporting of Software Diffusion (n=50)

Diffusion sub-types	Studies found
Shared ownership	2
Software distribution	8
Post-diffusion abandonment	1
Total studies reporting	9
No reporting	41

Diffusion was categorized into two areas: 1) evidence of shared ownership as diffusion, and 2) the software distribution itself. ‘Shared ownership’ referred to collaborative, open-source opportunities afforded by distributing software on the web. While the presence of diffusion as shared ownership was rare, the findings reflected the opportunities and affordances gained by diffusing software. Bug testing, submitting change requests (Branon, Wolfenstein, & Weiss, 2016), and applying open-source software licensing (Salas-Morera, Arauzo-Azofra, García-Hernández, Palomo-Romero, & Hervás-Martínez, 2013) encouraged software re-usage by facilitating additional iterations, forks, and updates by a wider community – even if there was no call-to-action to do so.

‘Software distribution’ was defined to either the explicit intent (post research) or action (during research) to provide the software for public access and usage. All instances of software distribution actions provided a web URL to locate the software or a project page listing the software, reflecting a coupling between the hosted software and the accompanying communications. Finally, a curious outlier study self-identified what can best be described as post-diffusion abandonment: evidence of software code stagnation and deprecation occurring after its diffusion, going so far as its authors referring to code as “withering” (Branon et al., 2016). This finding highlighted two barriers of software development not commonly considered: (1) software development best practices can move exceedingly quickly, and (2) writing good, solid software can require multiple iterations. Certain fields of software development are situated on shifting best-practices, toolkits, and development frameworks that historically stay in vogue for only a handful of years. For example, web development best-practices shifted significantly in the fifteen year timespan of analysis from platform- and system-centric to framework-centric development.

Additionally, software testing and refactoring was shown to be a time-intensive, exhaustive, and heavily-iterative process. Potential risks in the process of changing software not in its external behavior but its internal structure, or refactoring, included creating subtle software bugs, scope creep, and additional resources required to optimize code (Fowler, 2018). Refactoring would provide opportunities to develop easily-readable and -modifiable code for diffusion. The process of software testing, composed of software verification and validation, would require insight from domain knowledge experts who would validate the requirements, specifications, and artifacts of the created software (Myers, Badgett, Thomas, & Sandler, 2004). Software testing would be crucial to delivering validated, bug-minimized software.

However, there existed value in diffusion despite the barriers defined above. The choice of platforms, languages, or best-practices for software development did not have to be in vogue to be beneficial in authentic, real-world scenarios. Integrations of new technology in higher education commonly occurred as late-adopters. Positioning the Internet as a global and universal tool harnessed by learners and users across broad spectrums of socio-economic statuses, it would be naïve to assume that the global Internet audience moved in-step to platform changes, providing opportunities for longer shelf-lives for produced software from a global lens.

Software adoption.

50 studies were analyzed for presence of software adoption. Sub-types emerged from analysis to categorize and group similar adoption types and to categorize additional metadata. The results are described in Table 9 and Table 10.

Table 9

Software Adoption Presence (n=50)

Adoption sub-types	Studies found
Full Adoption of Software	8
Adoption of Additional Software	9
Evidence of after-development adoption	2
Total studies reporting	18
No reporting	32

Table 10

Software Adoption Sources (n=50)

Software sources	Studies found
Academic research	5
Software application	7
Software module	6
Total studies reporting	17
No reporting	33

Two studies self-identified post-development adoption, providing evidence of software portability and the speed-of-adoption from originating research into new projects. Software adoption evidence emerged into two sub-types: a full adoption of software, and adoption of software as part of research and development. Full adoption of software encompassed research repurposing existing software in its entirety. This perspective focused research lenses on instructional design related to, or analysis of, the pre-existing software, rather than software development itself. Discussion of software development adoption situated around its re-use would be valuable in confirming or validating the existing research and extending the scope or analysis of previous research, thereby extending the learning ecology within IDT.

Each study was coded to emergent synthetic constructed categories of software sourcing: as a product of previous academic research, general software product with no identified sourcing context, or a software module. Both attributes were valuable to discover the context to which adoption occurred and to define the types of adopted software, which were discussed in further detail with an additional layer of analysis.

Software adaptation.

50 studies were analyzed for presence of software adaptation. Sub-types emerged from analysis to categorize and group similar adaptation types. The results are described in Table 11.

Table 11

Software Adaptation presence (n=50)

Adaptation sub-types	Studies found
Software inspiration adaptation	5
Software module adaptation	6
Software application adaptation	5
Post-diffusion adaptation	0
Total studies reporting	15
No reporting	35

Software adaptation evidence categorized into three sub-types: adaptation of software, adaptation of software module into a larger product, and adaptation as inspiration: an adaptation of ideas, content, user-interface into a new context. A nearly parallel count of studies which self-reported adaptation (between software applications, software modules, and software inspiration) paired with a lack of overlap between adaptation sub-types: only one study provided evidence of more than one adaptation type. This provided evidence that research leveraging adaptation primarily focused on one aspect – repurposing of software, content, or a module. Evidence of studies extending multiple software types during development were not found except in a singular instance.

Inspiration serving as evidence of adaptation, relative to findings of adaptation of software and software modules, revealed a low-technology threshold available for research to identify and document software re-use. The inspiration evidence took two primary forms: content inspiration (instructional storylines or content) and development inspiration (high-level software overviews or goals). Both forms applied adaptation in that the self-identified inspiration directed and influenced project software development outside of the original scope of the referenced source. This aspect would be valuable for researchers that may be uncomfortable with more technically-lensed software analyses by self-identifying prior work influence and inspiration evidence. This evidence provided a bookmark for tracking the evolution of content or development inspiration through the application of adaptation, as well as crediting a previously-developed product for an attribute, aspect, of software worthy of adaptation. Unlike adaption and diffusion, no evidence of post-diffusion evidence of adaptation emerged. This finding was not early unanticipated, as it seemed unlikely that other research would locate the created software for adaptation prior to publication. Evidence of post-diffusion adaptation would presumably have a longer timeline than adoption and diffusion reporting, as adaptation would need to extend the software in new context or a new perspective and report back to the original developers before the research was complete.

Discussion

RQ1. How is software design and development detailed in IDT literature?

An initial round of inquiry detailed the scope of basic software development concepts and attributes within IDT research. Educational software platforms are well-reported (see Table 2), likely owing to the focus on learning outcomes within research. Gaps in software development self-reporting emerged as the technical requirements intertwined with the topic area increased, owing to the instructional research focus. This could be sourced from recommendations for software design and development to be reported within conference proceedings rather than journal articles. The current state of software design and development in IDT research educational software development was not well-reported for re-use. The primary method of analyzing development was through user-interface decisions - Table 5 highlighted the prevalence of interface- and graphic design. Only a few studies represented software development as high-level functionality (what does the software do?) and workflow (what major components, either hardware or software, does the developed software interact with?). Detail and analysis of software development was largely absent (Tables 4-6), with little emphasis or clarity on who developed, how the development occurred, and overall detail of the software's construction. An additional round of analysis (Tables 7-11) revealed the lack of software re-use evidence within existing literature using refined definitions of re-use components. The high number of undisclosed software developers (Table 6) strengthened the argument of Oliver (2011), in that the examination of technology in ILD would benefit from socially- and culturally-angled perspectives in order to examine technologies embedded within research, in order to understand the biases, influences, and perspectives that influenced the resulting software. As software developers were found to be unidentifiable (32) or outside the research team (10) compared to self-reporting of internal developers (30), more transparency and detail on the development process and team would aid the transfer of project and software development-focused knowledge both within the research team and to similar research.

Research has reiterated that individual programmer contributions (and their historical and cultural backgrounds) provided evidence for the functionality and design of the software (Shavelson, Phillips, Towne, & Feuer, 2005; Wang, Nieveen, & van den Akker, 2007). Knowing these areas of concern, it was pertinent to examine how the field of computer science has handled similar considerations. Computer science research has spent considerable resources evaluating and testing re-use and replication studies on software development, which routinely focused on a documentation plight. Kernebeck (1997) described ‘reusable’ computer code as a combinatory pairing of source code, documentation, and any aiding utilities. One potential avenue to help mediate this issue was embedding documentation within code comments, which would be written within the source programming code. However, code comment documentation remained a persistent and common problem within computer science, finding that the resulting documentation was out-of-date, broadly-focused, or inaccessible to other developers (Forward & Lethbridge, 2002; Marcus & Maletic, 2003; Moreno, 2014; Tan, Yuan, Krishna, & Zhou, 2007). Efforts to automate or generate documentation was a continual effort of research (McBurney & McMillan, 2014; Robillard, 1989). Computer science-focused research found that the broad queries of ‘why’ and ‘how’ we interpret computer source code persisted within both individual developers and teams (Ko, DeLine, & Venolia, 2007). A solution for IDT, then, would need to look to aid re-use and replication from a different perspective.

RQ2. How does software re-use occur within IDT literature?

Table 3 reviewed software re-use within the literature, locating 9.7% (7) studies which explicitly detailed software re-use. However, ILD’s decomposition of re-use provided a detailed frame-of-reference by which to analyze research, finding 40.2% (29) of studies reporting a re-use component in Table 7. The discrepancy in findings between the two phases highlighted a dichotomy within the field. Framing and detailing software development analysis and detail was non-standardized, which in turn made self-reporting software re-use difficult without pre-established consensus and best-practices. This finding revealed that 30% of studies (22) implicitly identified and detailed software re-use, whether the authors intended or not. The implicit nature of the detail revealed that knowledge applicable for re-use may be lost in this implicit declaration without clearer identifiers and benefits for framing analysis from software re-use. Table 7 highlighted the difficulty in reporting on re-use components defined by ILD, where 27% of studies (20) reported one component of re-use and only 12.5% of studies (9) reported two or more components.

RQ2.1. How does software diffusion occur?

12.5% of studies (9) reported software diffusion through distribution (11.1%) or distributed software ownership (2.7%), revealing the rarity of software sharing within the field when compared to the ubiquity of custom developed software (94.4%). Finding a sizeable presence of internal developers (41.6%) embedded within research projects revealed that projects already had a subject-matter expert that would be expected to have technical knowledge to diffuse the software onto the Internet. This finding revealed some difficulty in diffusing software and research projects and the best-practices and consensus for doing so. Additional detail and analysis within each study on how software was diffused for re-use could help establish best-practices for research to share software code and ownership. Diffusion could be impeded by buggy, unstable, or early software – and the hesitations in releasing software in that state. The distributed model of software diffusion, and shared ownership, provided an opportunity for other developers (and adoption or adaptation projects) to iterate the software and potentially fix those issues. There may be pre-existing notions of questionable value for sharing source code, due to difficulty in locating similarly-developed software (Hucka & Graham, 2018). Source code was not a reliable method of determining the functionality and design philosophies of software, but could be valuable in facilitating stricter software adoption and adaptation, thereby building consensus and best-practices through distributed, iterative software development practices. Adaptation could provide a suitable outlet for reconfiguring, customizing, and extending the source code into different contexts.

Following the prescriptions laid out in software searching mechanisms analyzed by Hucka and Graham (2018), the search and retrieval process for locating software was a need-driven endeavor across a number of sources including public software repositories, web searches, socially-focused help websites, mailing list discussion threads, social media communications, and software indexes. Diffusion would not need to facilitate the communications for potential searchers, but rather focus on making the developed software easy to locate. Additional written information would provide valuable context that can take the form of documentation, historical data, point-of-contacts, or software metadata. Just as important, however, would be that the same written information provide context clues to search engines and other web aggregator services to locate, index, and then serve the URL to visitors conducting searches using similar search terms. That is, the content accompanying the software would not only benefit those who locate the software, but also help maximize opportunities for discovery.

RQ2.2. How does software adoption occur?

Tables 9 and 10 found 25% of studies (18) reported software adoption of additional (12.5%) or the primary (11.1%) software. Comparatively, only 9.7% of studies self-identified code re-use (Table 3). This dichotomy, much like in RQ2, provided evidence for a lack of consistent definitions and frames-of-analysis for describing software in IDT. The findings of software sourced from academic research (6.9%), software packages (9.7%), and software tools (8.3%) revealed the breadth of software adoption, in that software both within (academic research) and outside (software packages and tools) IDT contributed to software development research. The identification and categorizations of software adoption sub-types – primary and additional software – revealed that software adoption occurred for entire software as well as individual software components. These findings, while a comparatively low percentage to the full breadth of custom development (94.4%) reviewed, provided real-world instances of software adoption as well as identification of sources to locate applicable software. This discrepancy revealed the incidental and ubiquitous nature of reporting software adoption, in that research may be subconsciously adopting software. Modern-day software development methodologies steered their design towards piecemeal software components. Web content-management systems, such as WordPress and Drupal, contained software module ecosystems acting much like LEGO bricks, where a developer could combine several modules into a functioning web application.

RQ2.3. How does software adaptation occur?

Table 11 found 20.8% of studies (15) reported software adaptation as inspiration (6.9%) or sourced from full software (8.3%) or a software tool (6.9%). This discrepancy provided additional evidence for the value in research oriented from perspectives of software development in developing software best-practices. This aspect was especially crucial in software adaptation to trace the growths and evolution of software between projects. Much like design-based research, these software projects would be guided by iterations, distributed through separate teams and project-goals. Rather than an upstream increase in complexity and fidelity between iterations, software adaptation would result in a series of software forks appearing more like a spider-web of derivations of the original software, branching into unanticipated scopes and directions. While this series of software forks would make it harder for iterative changes to scaffold back down to the original design, it instead would facilitate a library of similar software meeting a wider variation of needs. More importantly, each forked software would provide confirmation or rejection of a choice in a previous generation, routing best-practices through an evolutionary-like approach. However, this perspective would be reliant on wider usage of adaptation in IDT research, which was not evident in the findings.

A Perspective Forward

During the literature review, a recurring theme emerged: the placement of software as a tool with a singular role within IDT research. Research methodologies and tools centered analysis around instructional goals rather than a software creation processes. Some considerations outlined earlier, such as unstable and early software, were posited by their authors to be mitigated by additional development cycles and continued research. Pre-existing expertise in complimentary fields such as computer science, however, cannot be easily mitigated. Studies routinely glossed over, ignored, or excused any detail of the software development process, which an additional development cycle or continued research could not resolve. Missing analysis on the process or development could do additional harm to the software, as those underlying issues could be compounded through software refinement and additional cycles, which would increase the complexity and scope. Instead, a realistic and robust approach to re-orienting educational software development reporting in IDT needs to provide a methodological toolset analyzing research from the perspectives of the processes, decisions, and developers. A combination of socially-centered research methodologies highlighting software developers, and reporting tools highlighting detail and process, could provide software development process reporting for replication and re-use.

The literature review provided ample evidence to locate rationale for conducting research from the perspective of the software developers (see Table 5, Table 6). Software developers are the instrumental connection to translate instructional goals onto a wide range of platforms (Table 2), programming languages, and tools (Table 4) - and doing so with largely custom software development (Table 3). Therefore, it is imperative that IDT research captures the contributions, decisions, and negotiations of software developers within the research. A methodology combining design-based research (DBR) and cultural-historical activity theory (CHAT) serves to accomplish this perspective.

DBR’s original scope addressed the difficulty in software replication and re-use, focusing on cyclical design construction (a reversed complexity un-layering) where a project becomes more complex with each iteration. DBR research should provide sharable results to both practitioners and designers, and provide documentation that is aimed at study replication (The Design-Based Research Collective, 2003). Recursive development, defined by DBR to include both the educational software and its corresponding theoretical underpinnings, allowed for documentation creation embedded within research (Penuel, Fishman, Cheng, & Sabelli, 2011), centering the software as the subject of research. DBR focused design considerations to ease-of-adoption, dissemination, and reusability (Fishman, Marx, Blumenfeld, Krajcik, & Soloway, 2004), forcing considerations beyond the original project regarding scope, scale, and applicability (Veletsianos, Beth, Lin, & Russell, 2016) for early and unstable software.

DBR was expected to answer long-standing questions between software and research (Amiel & Reeves, 2008) by extending and solidifying a learning ecology - a combination tool-set of tasks, problems, tools, and implementations (Cobb et al., 2003). The emergent learning ecology would introduce a synthesized foundation of technological processes and best-practices (Amiel & Reeves, 2008) between similar software, providing a path towards consensus, best-practices, progress, and heuristics in educational software development. This utopia did not come to pass. Recent DBR research lacked re-use (Fishman et al., 2004). A wide majority of DBR software design projects, spanning multiple subject areas, only reported a single iteration (Ormel et al., 2012). Contrary to cyclical design concepts, DBR projects frequently did not report on the entirety of a research procedure (Vanderhoven, Schellens, Vanderlinde, & Valcke, 2016). Research highlighted, not addressed, perspective gaps between researchers and practitioners within a project (McKenney & Reeves, 2013). What DBR was able to accomplish was to provide a systemic methodology to scaffold software development – from humble planning, goals, to prototypes, and finally to software deliverable – through cycling stages of increasing complexity. Many of the resulting gains, such as replicability and reuse considerations, would trickle down from a more strict DBR interpretation.

Analysis of the research questions revealed that framing discussion, process, and detail about software development was hard. To properly leverage DBR, defined from its original scope, project analysis would need to be situated through a software developer perspective. How do IDT researchers then provide that perspective? Researchers could include computer science experts in each software development-focused research project. However, the review provided a cautionary case, as a study reported computer science experts describing the software development process as ‘trivial’ (Borro-Escribano et al., 2014). It would be very possible that a software application could be viewed by computer scientist experts as a trivial creation comparative to modern-day areas of research. Additionally, it would be unrealistic to expect a computer scientist to be embedded in every software-centered IDT project. Instead, IDT needs a process to talk about perspective gaps between experts in different fields. Those perspective gaps still would exist within a project regardless if a computer science expert is embedded. The value in these knowledge gaps, instead, would be how the contradictions in understanding differing perspectives is resolved; of how primary researchers, practitioners, graduate students, and external members make individual meaning of translations of instructional goals to software plans, and how those translations reconcile in a team-centric setting.

Those meaning-makings, and the contradictions that exist between those stakeholders, can be represented and documented using activity theory – specifically cultural-historical activity theory (CHAT). Activity theory’s original subject-tool-object triad, as defined by Leont'ev (1974) and Vygotsky (1978), was expanded to CHAT, which was composed of six components identifying societal influences on collaborative activity completion (subject, tools, rules, object, community, division of labor). This expanded analysis mapped the relationships of an activity’s goal between stakeholders and team members (Engeström, 1996). CHAT’s decomposition of an activity into its structural components and connections, including individual internalized social-, personal-, and historically-derived practices and assumptions (Hadjistassou, 2012), make it a powerful tool to examine group activities (Karasavvidis, 2009) like software development. CHAT facilitates the reporting of social decisions and processes intertwined within software development (Collis & Margaryan, 2004) through contradictions and change negotiations (Engeström, 1996). As with any software project, software decisions do not scope to only a singular level – it shifts from high-level data decisions to low-level programming code debate and discussion. CHAT accommodates focused analysis based on the scope and scale of the activity observed (Karanasios et al., 2013), with all activities constructing a joined activity system – an activity network (David & Victor, 2002). This activity network provides an negotiated-centric profile of the developed software. This allows for CHAT analysis to sway between scope and scale per activity, while also producing an overall roster of project contradictions and resolutions.

Limitations & Considerations

The professional and educational background in computer science of the primary author afforded a unique perspective for approaching the review of literature through CIS. The search, selection, categorization, coding, and quality-determination were influenced by experiences and expertise in software project management, open-source software community contributions, and variation between programming language features and functionality. A replication study leveraging the search and selection criterium would invariably produce varied categorization and coding methods by way of cultural, historical, and personal influences.

Future research could build upon the review of literature regarding software re-use in IDT. Replication studies could consider additional methods of determining database-friendly queries for locating developed software as a response to the ambiguity of terminologies for design and development between the fields of IDT and computer science (Oncu & Cakir, 2011). Future research could focus on the generation of synonym pairings for how software development is described by researchers in both computer science and IDT, in the creation of a shared syntax vocabulary which identifies the presence of software development. Future research could approach this topic from the perspective of the methodologies of the literature, as a wide majority of studies did not primarily focus on the development process of the software. A review of the literature on software maturity (preliminary versus well-iterated, or long-lived) would provide an additional perspective on software dissemination in the field.

Chapter 2

An Analysis of Process at the Nexus of Instructional and Software Design

The study presented in this chapter was the result of recommendations derived from a systematic review of literature on IDT software re-use presented in Chapter 1 of this manuscript. The intent of Chapter 1 was to uncover processes, strategies, and best-practices that might facilitate IDT software replication and re-use. An examination of a 15-year time period within four representative journals identified 72 studies that addressed IDT software to some extent. An additional sampling of the initial results identified 50 studies that discussed software development. These studies were further analyzed for evidence of software re-use. Review results found a lack of reusable and/or replication-focused reports of instructional software development in IDT journals, but found some reporting of IDT reuse and replication from articles outside of the field. Based on the analysis, possible reasons for this occurrence were discussed. The researcher then proposed how a model for conducting and presenting IDT research based on the constructs of design-based research (DBR) and cultural-historical activity theory (CHAT) might help mitigate this gap.

Rationale

Closing the knowledge gap uncovered by Sembrat (2020) is critical for a cross-disciplinary area of study such as IDT that finds itself at the nexus of computer science-driven perspectives of software design and development and the learning and instruction driven strategies and goals of IDT. Historically, IDT research has sacrificed substantial analysis of the technological creation process in lieu of heavy emphasis on its utilization (De Boeck & Minjeong, 2018), masking perspectives of the creation process that would be beneficial for researchers and practitioners who aim to extend similar research and development, and potentially leaving a steep adoption curve for each new research and development team without the advantage of educational software reusability and replicability. IDT software is not spontaneously or arbitrarily produced; rather, its creation is a combination of myriad choices and considerations (Oliver, 2011) that are embedded within individual and cultural decisions and contributions (Dennehy & Conboy, 2017). These projects are expensive to carry out, and they are too often constrained by a narrow selection of technologies that given research teams might be familiar or comfortable with, rather than what works best for project goals, thus greatly limiting the power of careful selection (Schuch, 2001). This goal of this study is to uncover strategies for IDT software design and development in a way that makes it more mutually understandable and replicable in a fashion similar to how the academic literature itself grows, that is, by building on the work of others. It is hoped that the design principles and framework introduced in this article – the Iterative and Integrative Instructional Software Design (IIISD) model - may inform those conducting IDT software design and development research as well as shine a light on reporting and analytical strategies for future similar projects.

Purpose

The qualitative, single-case study presented here incorporated Iterative Categorization (IC) for data analysis and Cultural-Historical Activity Theory (CHAT) as an analytical lens to examine a process of team-based software development within a large IDT design-based project. The units of analysis were the digital communicative artifacts generated from the collaborative creation of a web software application for supporting middle school students' learning how to code in an urban after-school program. Analysis involved deconstructing social decisions and processes intertwined within the software development process in question (Collis & Margaryan, 2004) by uncovering contradictions and mediations (Engeström, 1996). The deconstruction depicted in this chapter mapped one strand of the software creation process, the development of an online badging system, through the activities of various subject-matter experts at pivotal project points. The study was guided by the following research question:

- How did social contradictions and mediations shape the software development process within a large educational research project?

Analytical Frame

Iterative Categorization

Founded in addiction-focused qualitative studies within health sciences (Plummer, 2018), IC is a data analysis method that focuses on the reduction of raw observational data into data visualizations for broad theme-focused interpretations through analyses at three levels: descriptive, interpretive, and summative (Neale, 2016). Prior to data analysis, researchers construct a coding book and frame. Each piece of data is coded through a ‘systematic order and sort’ method outlined by Neale (2016). In this method, deductive codes are derived from observed topic areas in collected data, which are placed under broadly-defined, general headings. Then, inductive codes are appended and existing codes are supplemented based on emergent findings. Next, researchers conduct the three levels of analysis.

In the first level of analysis, descriptive analysis, researchers select pertinent coding frame categories. Copies of the coding frame Word document file are created for each category.

Researchers then record topic identifications, themes, conflicts between other codes, applicable quotations and diagrams for every piece of data within the coding book. Additionally, researchers may record mental observations such as spontaneous thoughts, mind maps, and ideas from the consumption and analysis of data. This phase of coding produces frame-specific coding books along with self-reported observations. In the second level of analysis, interpretive analysis, researchers identify associations, explanations, and patterns within the data for each descriptive analysis category. The mental observations noted in the descriptive analysis are tested and expanded for applicability and validity. Additionally, identification and exploration of data and their relationships to common assumptions in IDT are examined, referencing additional literature, theory, or practices. This phase produces truncated coding books focused on connecting data points and identifying and sourcing patterns and outliers. The resulting document also produces supplemental data and knowledge to respond to researcher observations. Finally, summary analysis constructs executive reports of descriptive and interpretive analyses for the selected codes from the coding frame. These transparent and repeatable translations maximize external validity for analysis conducted by a single researcher (Neale, 2016).

Activity Theory

Founded in the works of German philosophers (Kant and Hegel), the works of Marx and Engels, and Soviet Russian cultural-historical psychology (e.g., Vygotsky, Leont'ev, and Luria), activity theory (AT) is a lens for the analysis and interpretation of a human behavior process, such as learning, through the lens of a social activity (Engestrom, Miettinen, & Punamaki, 1999). An activity is a long-term project directed towards transformation of an object (Matthews, Rattenbury, & Carter, 2007) through multiple, interconnected actions (Peña-Ayala, Sossa, &

Méndez, 2014). Cultural-Historical Activity Theory (CHAT) is a third iteration of AT, permutated to model a social activity as an activity system of interacting, interconnected environmental components towards completion to an outcome (Engeström, 1987). Historically- and culturally-formed artifacts shape the actions, both conscious and subconscious, which compose an activity (Postholm, 2015). The model activity system for CHAT contains six components by which to examine those artifacts. See Figure 2. *Subjects* are individuals (or groups of individuals) involved in an activity. *Objects* are the directions, sharable materials to be transformed or modified by the participants in, or abstractions (plans or ideas) of an activity. *Instruments* are anything utilized to help to carry out activities or to serve as models or experiences (physical objects such as computers and pens, psychological concepts such as languages or ideas). *Rules* govern social behaviors of community members within an activity (customs, relationships, or processes). *Division* of labor are the roles and responsibilities of subjects within an activity. *Community* are the people interacting within the environment of the activity (Choi & Kang, 2010). This study followed the philosophical positioning of CHAT analysis as defined by Sannino and Engeström (2018).

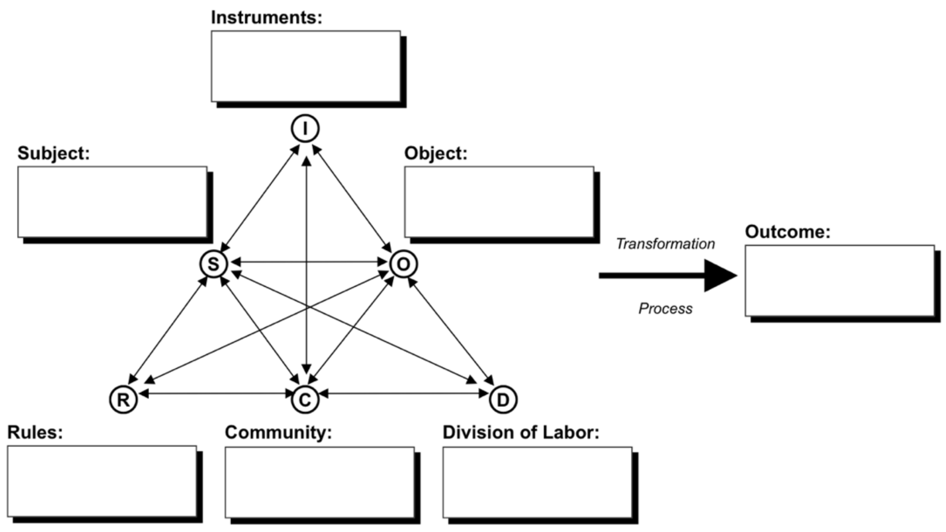


Figure 2. CHAT model (Engeström, 1987).

These components aid in identifying contradictions, which are the breakdowns, problems, tensions, or conflicts between components and within subsystems. Identified contradictions are categorized as resolved, in that a consensus is reached through mediation; or unresolved, meaning the contradiction persists in the end-result of the activity. Contradictions and mediations are categorized by their activity scope: within elements within a single activity system between roles within the same component (primary), between roles with differing components (secondary), an evolution of an activity system over time (tertiary), or within an interconnected combination of activity systems – an activity network (quaternary) (Engeström, 1987). These permutations allow visualizations and mapping of activity system evolutions and action motivations (Barab, Barnett, Yamagata-Lynch, Squire, & Keating, 2002), which can become complex and concentric. Activity systems can be analyzed and permutated from alternate or multiple perspectives throughout a single project, shifting from primary, secondary, tertiary, or quaternary scopes based on the activity analyzed (Barab et al., 2002). Nested activities and corresponding actions can commonly be expanded into their own separate activity systems (Korpela, Mursu, & Soriyan, 2002), owing to the underlying complexity and scaffolding of social activity.

Activity systems have been utilized across educational disciplines to better understand activities - digital records (Karanasios et al., 2013), information systems (Iyamu & Shaanika, 2019), instructional activities (Çakıroğlu, Kokoç, Kol, & Turan, 2016), lesson design (Lewin, Cranmer, & McNicol, 2018), professional peer knowledge sharing (Trust, 2017), serious games analysis and design (Carvalho et al., 2015), system design (Korpela et al., 2002), and web application development (Uden, Valderas, & Pastor, 2008). CHAT's flexibility and fluidity allows activity systems to more accurately reflect the inherent complexity and hierarchy embedded within an activity, such as internalized social-, personal-, and historically-derived practices and assumptions (Hadjistassou, 2012), in examinations of group activities (Karasavvidis, 2010) such as software development.

CHAT's flexibility to focus and variate analysis based on the scope and scale of the activity observed allows for contradiction identification on a macro- or micro-level (Karanasios et al., 2013). These activity-scoped analyses can be grouped together into an encompassed activity system (David & Victor, 2002) or an activity network (Peña-Ayala et al., 2014), depicting the overall activity and allowing for segmented deeper dives into micro-activities. This fluidity allows for the contributors, decisions, and perspectives, which have shaped the software's development (Panke et al., 2007) to be reflected within the activity systems, potentially extending IDT software analysis beyond merely being a vehicle for instruction (Dennehy & Conboy, 2017), and thus potentially making it easier for future similar projects to examine, incorporate, and replicate what has been done in the past (Lamb & Johnson, 2006). Doing this could resolve the issue pointed out by Carvalho et al. (2015), that existing instructional models, frameworks, and methodologies "do not fully answer the question on how the concrete components of the [system] have to be structured to support learning" (p. 166).

Method

Researcher's Position

The identification of the lead researcher's position followed formatting of Wade-Jaimes, Cohen, and Calandra (2019). The lead researcher for this project is a middle-class, White male working full-time as a web developer for a neighboring higher-education institution. He brought two cultural and historical perspectives to the project. First, he carried historical experiences of similar and previous projects, programming language usage, and acquired best practices as part of his work history. His historical knowledge lensed best-practices from two fields of study: he completed a computer science undergraduate program, and he was an active doctoral IDT student. Second, he carried cultural perspectives positioned solely as a subject-matter expert team member within the project. He was not an active team member in non-technical project meetings, project implementations, subject selection, or instructional design. The project team positioned his role and presence as a mediator between instructional project team considerations and the end-result technological intervention. Data analysis was positioned from his dual-lens understandings of computer science and IDT. The lead researcher is referred onwards as the 'web developer'.

Context

This study was embedded within a design-based research (DBR) project that took place at a public research university in a large, southeastern city in the United States. The project included the creation, implementation, and evaluation of a unique, technology-rich, informal learning environment for middle school youth. Participants were middle school students from 10 schools in the an inner-city public school system in the southeastern United States who were participating in a free, after-school program for roughly 2600 students at the time this study took place. The intervention involved participants working on computer science activities in media centers, computer labs, classrooms, and online. Activities consisted of: a) A guided series of programming steps that led to the creation of pre-designed mobile apps using a block-based interface; b) more gently guided problem-based tasks that allowed participants to continue working on, tweak, troubleshoot, or remix existing mobile apps; and c) opportunities for participants to work on their own original ideas for mobile apps.

Activities were presented to participants in a flexible, modularized fashion, and based on gradual increases of difficulty. Participants could submit completed activities in exchange for new activities of their choice, and for digital badges called *experience coins*. Activity submission, coin redemption, communication with others, and access to digital resources were all possible within a custom-built web software leveraging the Drupal content management system.

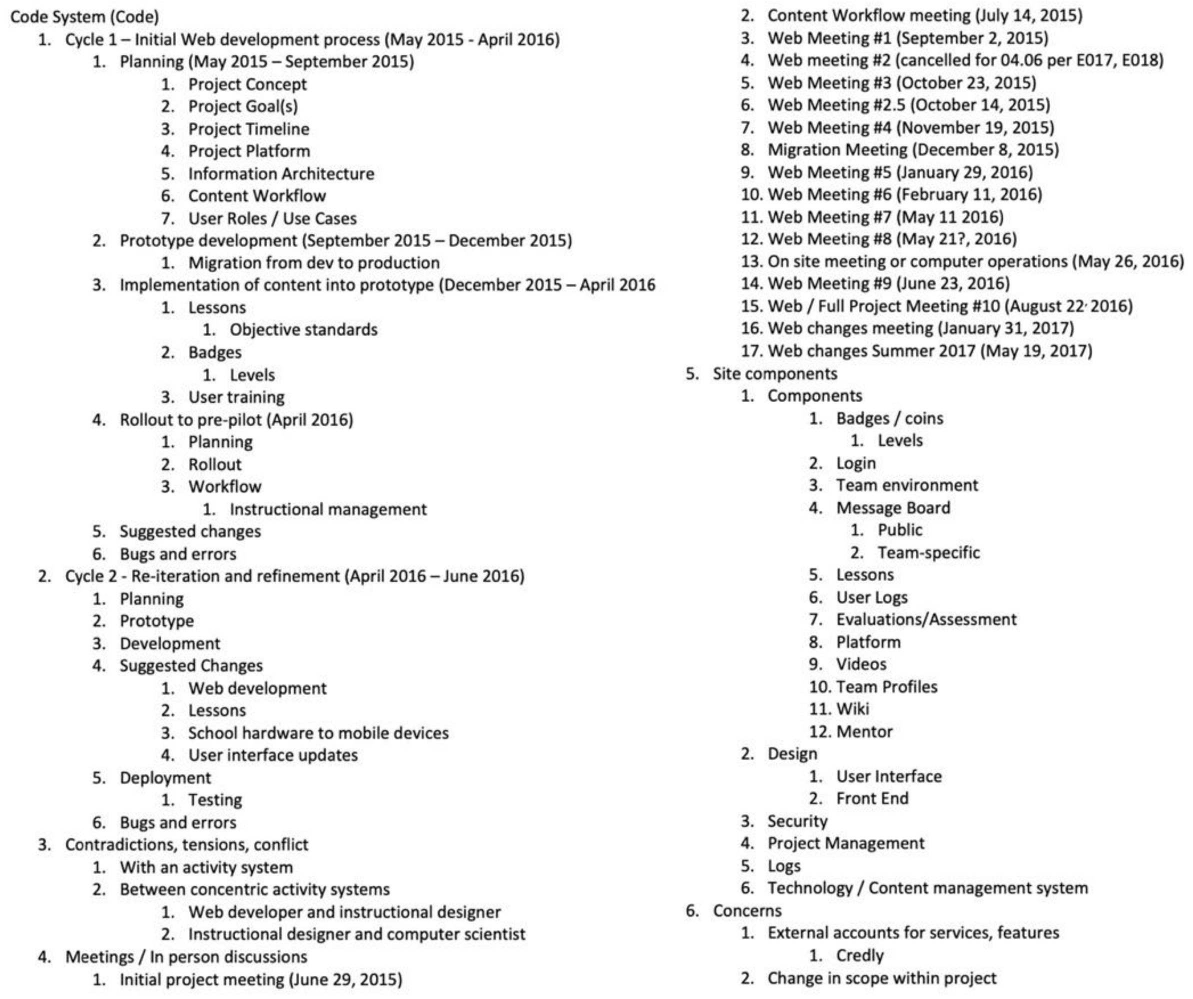
The research team was composed of three faculty members and two research assistants. One research assistant served as the web software instructional designer, creating lessons and activities. The second research assistant served as the web developer subject-matter-expert and was in charge of all web development. The research and development process involved four iterations: initial development, pilot study, beta test, and production phases. The current study is embedded within the project's initial development phase and pilot study phases, which were conducted in one of the middle schools. The web software creation process depicted here lasted 19 months.

Data Collection

This study involved the collection of qualitative data related to the instructional technology project management process of the project described in the previous section. The primary format of data collected was digital communications surrounding the development of a piece of IDT software. Data sources included email communications, documents gathered from Basecamp (online project management software), and notes plus photo captures of whiteboard diagrams from in-person meetings. Email was purposed for one-to-one communication. Basecamp was purposed for full research-team communication. A digital archive of all Basecamp pages including tasks, file uploads, and comments was created. All project emails on which the web developer was included were exported. For in-person meetings, both physical notes and mobile phone screenshots of whiteboard drawings were utilized. These data were then embedded into Basecamp for dissemination and storage. Collected data included 156 email threads (which contained 27 email-attached files), 164 tasks created in the online project management software application Basecamp (which contained discussion threads from team members), and 165 file attachment uploads attached to project tasks.

Data Analysis

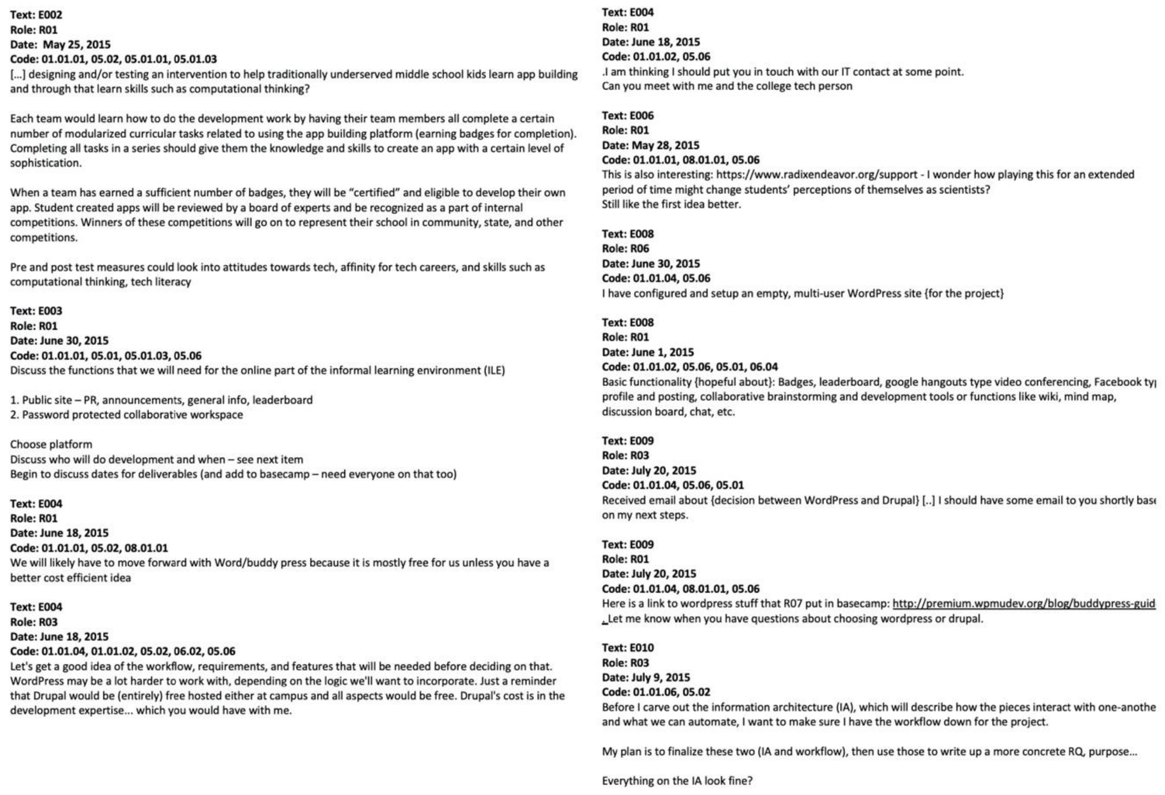
The researcher analyzed all qualitative data following guidelines from IC for categorization of the data and using CHAT as a lens for further analysis and presentation. Data were anonymized and categorized in order to aid coding and reporting. This resulted in single-letter aliases associated with each qualitative data type. Document data were renamed to D001-D027, emails from E001-E156, Basecamp tasks from T001-T164, and Basecamp uploads from U001-U165. Data referencing a team member were anonymized similarly to R01-R13. Using the newly labeled data, the researcher established a coding book and analytical frame. All data were coded leveraging the ‘systematic order and sort’ method borrowed from Neale (2016). Deductive coding was first used to established each of multiple cycles of software development: Planning, prototype, development, review and suggested changes, deployment, and debugging. More general labels were then created that were related to design and development processes: site components, concerns/problems, conflicts/contradictions. Next, the inductive codes were appended to the coding frame. During this process, the researcher categorized and ordered all project meetings, deconstructed the web software into components-of-focus, constructed a timeline for Basecamp tasks and their interconnections, and anonymized project team member codes for all corresponding roles. Additionally, inductive coding provided subcategories for deductive code placeholders. All data were collated into a coding frame. See Figure 3 for a sample of the coding frame. Data applicable to the software development process were copied into a code book. The resulting coding book contained 182 pages. See Figure 4 for a sample from the code book.



Code System (Code)

1. Cycle 1 – Initial Web development process (May 2015 - April 2016)
    1. Planning (May 2015 – September 2015)
        1. Project Concept
        2. Project Goal(s)
        3. Project Timeline
        4. Project Platform
        5. Information Architecture
        6. Content Workflow
        7. User Roles / Use Cases
    2. Prototype development (September 2015 – December 2015)
        1. Migration from dev to production
    3. Implementation of content into prototype (December 2015 – April 2016
        1. Lessons
            1. Objective standards
        2. Badges
            1. Levels
        3. User training
    4. Rollout to pre-pilot (April 2016)
        1. Planning
        2. Rollout
        3. Workflow
            1. Instructional management
    5. Suggested changes
    6. Bugs and errors
2. Cycle 2 - Re-iteration and refinement (April 2016 – June 2016)
    1. Planning
    2. Prototype
    3. Development
    4. Suggested Changes
        1. Web development
        2. Lessons
        3. School hardware to mobile devices
        4. User interface updates
    5. Deployment
        1. Testing
    6. Bugs and errors
3. Contradictions, tensions, conflict
    1. With an activity system
    2. Between concentric activity systems
        1. Web developer and instructional designer
        2. Instructional designer and computer scientist
4. Meetings / In person discussions
    1. Initial project meeting (June 29, 2015)
    2. Content Workflow meeting (July 14, 2015)
    3. Web Meeting #1 (September 2, 2015)
    4. Web meeting #2 (cancelled for 04.06 per E017, E018)
    5. Web Meeting #3 (October 23, 2015)
    6. Web Meeting #2.5 (October 14, 2015)
    7. Web Meeting #4 (November 19, 2015)
    8. Migration Meeting (December 8, 2015)
    9. Web Meeting #5 (January 29, 2016)
    10. Web Meeting #6 (February 11, 2016)
    11. Web Meeting #7 (May 11 2016)
    12. Web Meeting #8 (May 21?, 2016)
    13. On site meeting or computer operations (May 26, 2016)
    14. Web Meeting #9 (June 23, 2016)
    15. Web / Full Project Meeting #10 (August 22' 2016)
    16. Web changes meeting (January 31, 2017)
    17. Web changes Summer 2017 (May 19, 2017)
5. Site components
    1. Components
        1. Badges / coins
            1. Levels
        2. Login
        3. Team environment
        4. Message Board
            1. Public
            2. Team-specific
        5. Lessons
        6. User Logs
        7. Evaluations/Assessment
        8. Platform
        9. Videos
        10. Team Profiles
        11. Wiki
        12. Mentor
    2. Design
        1. User Interface
        2. Front End
    3. Security
    4. Project Management
    5. Logs
    6. Technology / Content management system
6. Concerns
    1. External accounts for services, features
        1. Credly
    2. Change in scope within project

Figure 3. Coding frame sample pages.



Text: E002
Role: R01
Date: May 25, 2015
Code: 01.01.01, 05.02, 05.01.01, 05.01.03
[...] designing and/or testing an intervention to help traditionally underserved middle school kids learn app building and through that learn skills such as computational thinking?

Each team would learn how to do the development work by having their team members all complete a certain number of modularized curricular tasks related to using the app building platform (earning badges for completion). Completing all tasks in a series should give them the knowledge and skills to create an app with a certain level of sophistication.

When a team has earned a sufficient number of badges, they will be "certified" and eligible to develop their own app. Student created apps will be reviewed by a board of experts and be recognized as a part of internal competitions. Winners of these competitions will go on to represent their school in community, state, and other competitions.

Pre and post test measures could look into attitudes towards tech, affinity for tech careers, and skills such as computational thinking, tech literacy

Text: E003
Role: R01
Date: June 30, 2015
Code: 01.01.01, 05.01, 05.01.03, 05.06
Discuss the functions that we will need for the online part of the informal learning environment (ILE)

1. Public site – PR, announcements, general info, leaderboard
2. Password protected collaborative workspace

Choose platform
Discuss who will do development and when – see next item
Begin to discuss dates for deliverables (and add to basecamp – need everyone on that too)

Text: E004
Role: R01
Date: June 18, 2015
Code: 01.01.01, 05.02, 08.01.01
We will likely have to move forward with Word/buddy press because it is mostly free for us unless you have a better cost efficient idea

Text: E004
Role: R03
Date: June 18, 2015
Code: 01.01.04, 01.01.02, 05.02, 06.02, 05.06
Let's get a good idea of the workflow, requirements, and features that will be needed before deciding on that. WordPress may be a lot harder to work with, depending on the logic we'll want to incorporate. Just a reminder that Drupal would be (entirely) free hosted either at campus and all aspects would be free. Drupal's cost is in the development expertise... which you would have with me.

Text: E004
Role: R01
Date: June 18, 2015
Code: 01.01.02, 05.06
.I am thinking I should put you in touch with our IT contact at some point.
Can you meet with me and the college tech person

Text: E006
Role: R01
Date: May 28, 2015
Code: 01.01.01, 08.01.01, 05.06
This is also interesting: https://www.radixendeavor.org/support - I wonder how playing this for an extended period of time might change students' perceptions of themselves as scientists?
Still like the first idea better.

Text: E008
Role: R06
Date: June 30, 2015
Code: 01.01.04, 05.06
I have configured and setup an empty, multi-user WordPress site {for the project}

Text: E008
Role: R01
Date: June 1, 2015
Code: 01.01.02, 05.06, 05.01, 06.04
Basic functionality {hopeful about}: Badges, leaderboard, google hangouts type video conferencing, Facebook ty
profile and posting, collaborative brainstorming and development tools or functions like wiki, mind map, discussion board, chat, etc.

Text: E009
Role: R03
Date: July 20, 2015
Code: 01.01.04, 05.06, 05.01
Received email about {decision between WordPress and Drupal} [..] I should have some email to you shortly bas
on my next steps.

Text: E009
Role: R01
Date: July 20, 2015
Code: 01.01.04, 08.01.01, 05.06
Here is a link to wordpress stuff that R07 put in basecamp: http://premium.wpmudev.org/blog/buddypress-guid
_Let me know when you have questions about choosing wordpress or drupal.

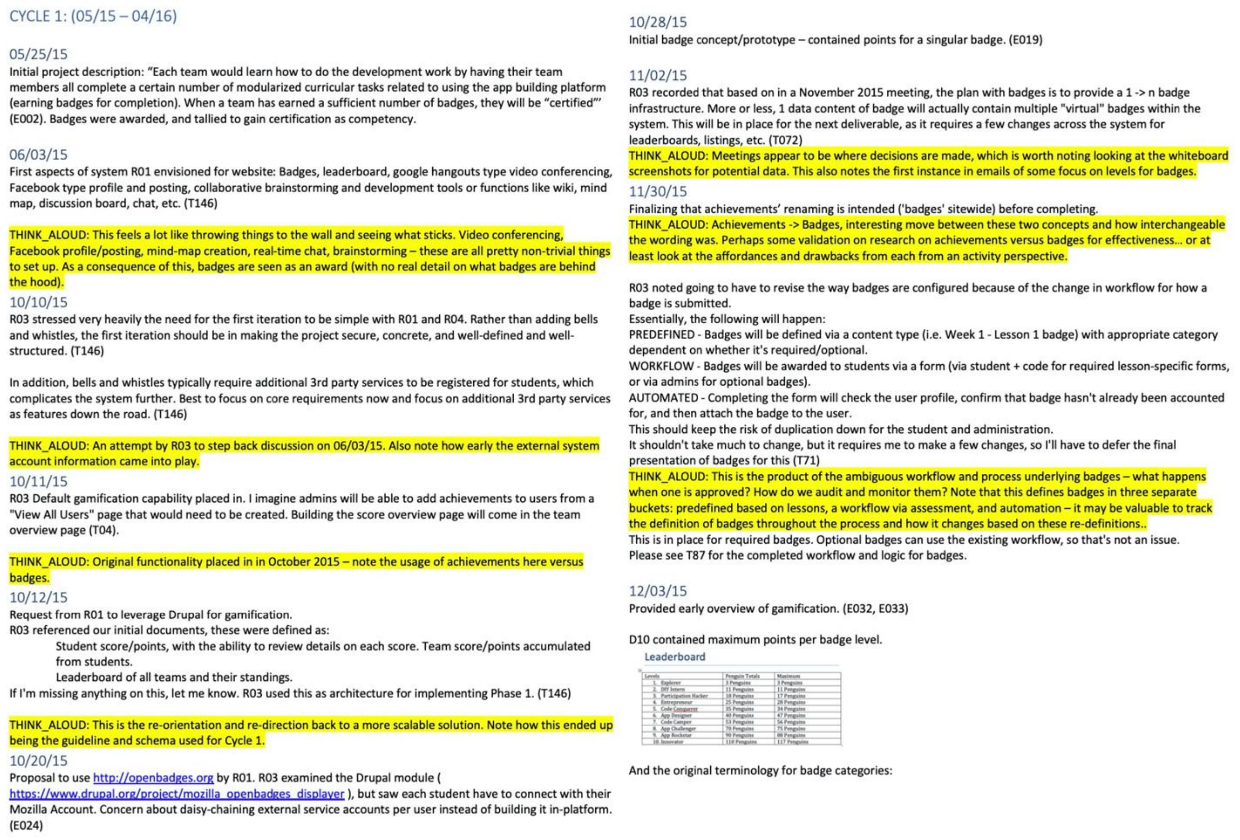
Text: E010
Role: R03
Date: July 9, 2015
Code: 01.01.06, 05.02
Before I carve out the information architecture (IA), which will describe how the pieces interact with one-anothe
and what we can automate, I want to make sure I have the workflow down for the project.

My plan is to finalize these two (IA and workflow), then use those to write up a more concrete RQ, purpose...

Everything on the IA look fine?

Figure 4. Coding book sample pages.

Next, the researcher examined the coding frame for elements relevant to the software development process in question. During this process, the implementation of badging emerged as a prominent design aspect of the software within each of the four development cycles, thus providing an exemplary focus representative of the design and development process. Data analysis identified multiple instances of contradictions and mediations related to digital badges throughout the web software development, providing sufficient insight into the general software development processes during the initial project cycles. The researcher next conducted descriptive analysis of the coding book focusing on the badging function of the software (frame code 05.01.01). A sample of the descriptive analysis is shown in Figure 5.



CYCLE 1: (05/15 – 04/16)

05/25/15
Initial project description: "Each team would learn how to do the development work by having their team members all complete a certain number of modularized curricular tasks related to using the app building platform (earning badges for completion). When a team has earned a sufficient number of badges, they will be "certified"' (E002). Badges were awarded, and tallied to gain certification as competency.

06/03/15
First aspects of system R01 envisioned for website: Badges, leaderboard, google hangouts type video conferencing, Facebook type profile and posting, collaborative brainstorming and development tools or functions like wiki, mind map, discussion board, chat, etc. (T146)

THINK_ALOUD: This feels a lot like throwing things to the wall and seeing what sticks. Video conferencing, Facebook profile/posting, mind-map creation, real-time chat, brainstorming – these are all pretty non-trivial things to set up. As a consequence of this, badges are seen as an award (with no real detail on what badges are behind the hood).

10/10/15
R03 stressed very heavily the need for the first iteration to be simple with R01 and R04. Rather than adding bells and whistles, the first iteration should be in making the project secure, concrete, and well-defined and well-structured. (T146)

In addition, bells and whistles typically require additional 3rd party services to be registered for students, which complicates the system further. Best to focus on core requirements now and focus on additional 3rd party services as features down the road. (T146)

THINK_ALOUD: An attempt by R03 to step back discussion on 06/03/15. Also note how early the external system account information came into play.

10/11/15
R03 Default gamification capability placed in. I imagine admins will be able to add achievements to users from a "View All Users" page that would need to be created. Building the score overview page will come in the team overview page (T04).

THINK_ALOUD: Original functionality placed in in October 2015 – note the usage of achievements here versus badges.

10/12/15
Request from R01 to leverage Drupal for gamification.
R03 referenced our initial documents, these were defined as:

Student score/points, with the ability to review details on each score. Team score/points accumulated from students.
Leaderboard of all teams and their standings.

If I'm missing anything on this, let me know. R03 used this as architecture for implementing Phase 1. (T146)

THINK_ALOUD: This is the re-orientation and re-direction back to a more scalable solution. Note how this ended up being the guideline and schema used for Cycle 1.

10/20/15
Proposal to use http://openbadges.org by R01. R03 examined the Drupal module (https://www.drupal.org/project/mozilla_openbadges_displayer), but saw each student have to connect with their Mozilla Account. Concern about daisy-chaining external service accounts per user instead of building it in-platform. (E024)

10/28/15
Initial badge concept/prototype – contained points for a singular badge. (E019)

11/02/15
R03 recorded that based on in a November 2015 meeting, the plan with badges is to provide a 1 -> n badge infrastructure. More or less, 1 data content of badge will actually contain multiple "virtual" badges within the system. This will be in place for the next deliverable, as it requires a few changes across the system for leaderboards, listings, etc. (T072)
THINK_ALOUD: Meetings appear to be where decisions are made, which is worth noting looking at the whiteboard screenshots for potential data. This also notes the first instance in emails of some focus on levels for badges.

11/30/15
Finalizing that achievements' renaming is intended ('badges' sitewide) before completing.
THINK_ALOUD: Achievements -> Badges, interesting move between these two concepts and how interchangeable the wording was. Perhaps some validation on research on achievements versus badges for effectiveness… or at least look at the affordances and drawbacks from each from an activity perspective.

R03 noted going to have to revise the way badges are configured because of the change in workflow for how a badge is submitted.
Essentially, the following will happen:
PREDEFINED - Badges will be defined via a content type (i.e. Week 1 - Lesson 1 badge) with appropriate category dependent on whether it's required/optional.
WORKFLOW - Badges will be awarded to students via a form (via student + code for required lesson-specific forms, or via admins for optional badges).
AUTOMATED - Completing the form will check the user profile, confirm that badge hasn't already been accounted for, and then attach the badge to the user.
This should keep the risk of duplication down for the student and administration.
It shouldn't take much to change, but it requires me to make a few changes, so I'll have to defer the final presentation of badges for this (T71)
THINK_ALOUD: This is the product of the ambiguous workflow and process underlying badges – what happens when one is approved? How do we audit and monitor them? Note that this defines badges in three separate buckets: predefined based on lessons, a workflow via assessment, and automation – it may be valuable to track the definition of badges throughout the process and how it changes based on these re-definitions..
This is in place for required badges. Optional badges can use the existing workflow, so that's not an issue.
Please see T87 for the completed workflow and logic for badges.

12/03/15
Provided early overview of gamification. (E032, E033)

D10 contained maximum points per badge level.

Leaderboard

Levels	Penguin Totals	Maximum
1. Explorer	3 Penguins	3 Penguins
2. DIY Intern	11 Penguins	11 Penguins
3. Participation Hacker	18 Penguins	17 Penguins
4. Entrepreneur	25 Penguins	28 Penguins
5. Code Conquerer	35 Penguins	34 Penguins
6. App Designer	40 Penguins	47 Penguins
7. Code Camper	53 Penguins	56 Penguins
8. App Challenger	70 Penguins	75 Penguins
9. App Rockstar	90 Penguins	88 Penguins
10. Innovator	110 Penguins	117 Penguins

And the original terminology for badge categories:

Figure 5. Descriptive analysis sample pages focused on badging.

The researcher made two deviations from the original scope of descriptive analysis to accommodate the data. Unlike the IC process reported by Neale (2016), where interview data was generalized into common themes, qualitative data within this study were categorized contextually, and related to their creation date. The presence of the date-of-creation provided context by which to map the qualitative data pieces to larger activities. In order to understand the procedural steps taken in between dates of qualitative data, the researcher organized the data chronologically after an initial round of descriptive summaries and think-aloud notes. The researcher also extended IC by conducting an additional, subsequent round of analysis after the chronological reordering in order to confirm and extend think-aloud notes, as well as to combine date-equivalent descriptive summations. The researcher defined this additional process as *chronological analysis*: A descriptive analysis iteration implemented when qualitative data is primarily organized and correlated chronologically.

Additionally, the researcher kept individual pieces of data intact in order to better identify emergent narratives, rather than grouping them into common generalized themes with qualifiers as outlined in the original scope of IC. This consideration facilitated the identification of the social contradictions and mediations during interpretive analysis. This consideration also maximized the presence and role of dialogue within AT to inform context (Wells, 2002). To aid the identification of researcher comments for interpretive analysis, the researcher placed visual identifiers on think-aloud notes (highlighted yellow and appended with "THINK_ALOUD" text) and identified candidates within the data for social contradictions and mediations (highlighted purple). This study applied the think aloud concept from the synthesized definition by Birch and Whitehead (2020), which captured problem-solving and decision-making data situated as in-event cognitions through immediate verbal reporting. Think aloud notes served as qualitative reporting of the mental observations (spontaneous thoughts, mind maps, and ideas from the consumption and analysis of data) by the web developer, as defined by Neale (2016).

Interpretive analysis was reported below as a series of activity systems driving the consecutive design and development of badging within the web software system. CHAT facilitated the "redirect[ion of] our gaze from what is going on inside the individual to what happens between human beings, their objects, and their instruments when they pursue and change their purposeful collective activities […] by means of cultural artifacts" (Sannino & Engeström, 2018). CHAT's positioning of a blended cultural and historical lens is best illustrated by an analysis of janitorial cleaning summarized by Sannino and Engeström (2018), where vacuum cleaning methods were subconsciously influenced both by their past frequent at-home mopping activities (historical) and social, ritual home cleaning instilled by European culture and propaganda as an economic status symbol (cultural). This positionality facilitated analysis which could explore how and where intersections between a web developer with a computer science background – with its embedded cultural (software development culture, best-practices) and historical (precedent conscious or unconscious habits or on previous experiences) – shaped an instructional design request towards its enaction and completion.

Analysis employed CHAT to decompose and visualize potential contradictions and mediations identified within data in order to better understand tensions between the two distinct instructional design and software development design perspectives. Analysis distilled activity systems into broad-scoped activities in order to facilitate combinations of primary and secondary contradictions (Postholm, 2015). Analysis constrained scope to primary and secondary contradictions to minimize the complexity of the activity systems. The researcher depicted contradictions as two-headed lightning shaped arrows, similar to Engeström (2000). See Figure 6. The researcher used two-headed lightly-shaded arrows to depict mediated contradictions. Unresolved contradictions were depicted as two-headed lightning shaped arrows with dashes. Additionally, primary contradictions were depicted as circular arrows mirroring the same criterion above.

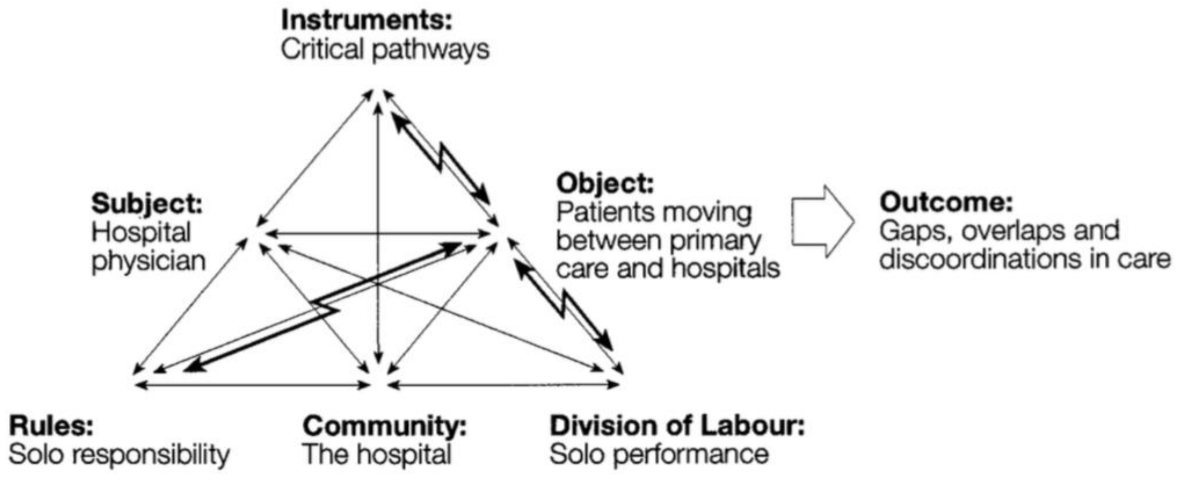


Figure 6. Secondary contradiction and mediation visualizations in the activity triangle model (Engeström, 2000).

Ambiguous definitions of 'mental models' (Kankuzi & Sajaniemi, 2016) facilitated the creation of a bounding definition to help frame contradiction mediation. In this definition, mental models reflected a user's understanding of three components of a real object from three perspectives: what it contains, how it works, and why it works that way (Carroll & Olson, 1987). Additionally, the definition positioned these models as a specialized subset of an individual's entire knowledge on a subject framed by a self-identified perspective of appropriateness (Kankuzi & Sajaniemi, 2016). Mental models situated from the web developer subject, as the primary author, were more accurate reflections of the specialized subset of knowledge for the activity. Mental models situated within activities for other subject roles were interpreted by the primary author from existing qualitative data that contained impressions of cultural and historical perspectives. No additional accommodations were made to extend or validate the interpreted impressions, as the study was situated from retrospective analysis after both the completion of the project and creation of all collected data.

Interpretive analysis uncovered four major areas of design and development negotiations that were decomposed as activity systems: initial instructional design, gamification as ecommerce, badges and team profiles, and qualitative assessment review. See Figure 7 for a representation of all uncovered contradictions and their formats.

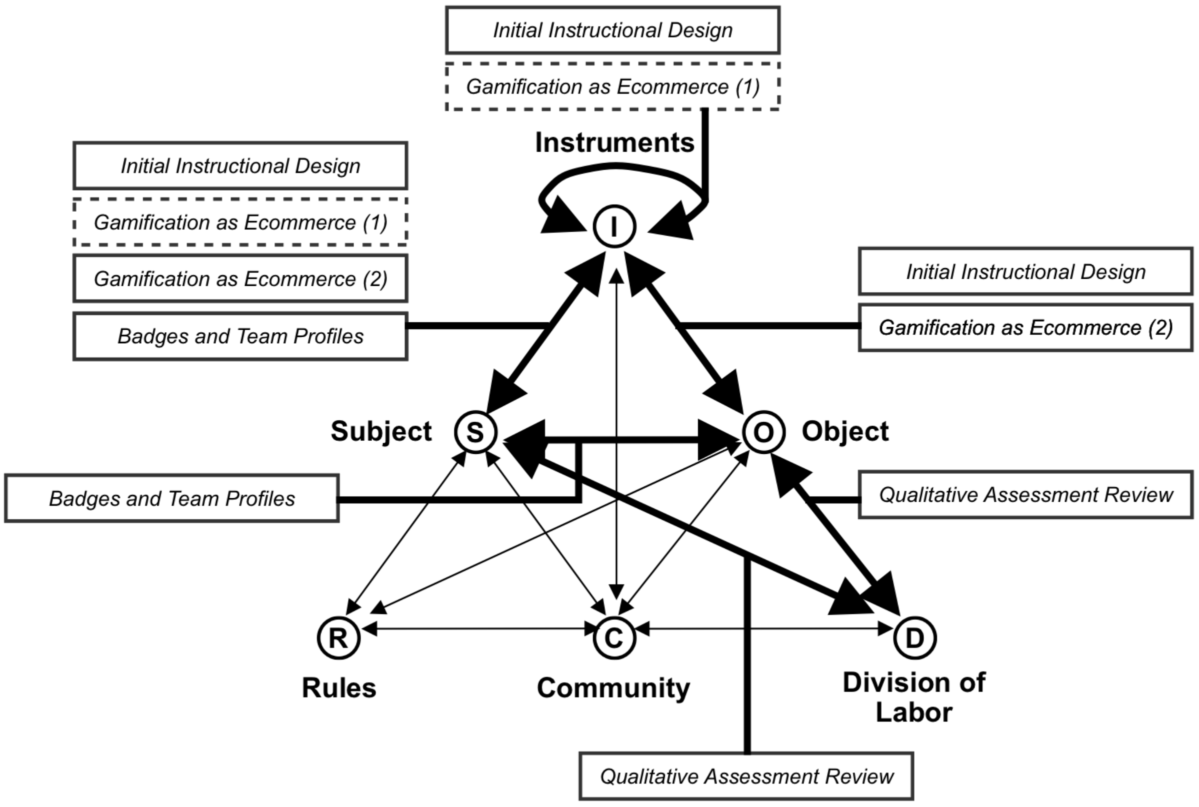

Initial Instructional Design
Gamification as Ecommerce (1)
Instruments
Initial Instructional Design
Gamification as Ecommerce (1)
Gamification as Ecommerce (2)
Badges and Team Profiles
I
Initial Instructional Design
Gamification as Ecommerce (2)
Subject
S
O
Object
Badges and Team Profiles
Qualitative Assessment Review
R
C
D
Rules
Community
Division of Labor
Qualitative Assessment Review

Figure 7. All identified contradictions and mediations.

Initial instructional design.

During development cycle 1, the faculty members and web developer developed an initial web software prototype guided by the project description: "…each team would learn how to do the development work by having their team members all complete a certain number of modularized curricular tasks related to using the app building platform (earning badges for completion). When a team has earned a sufficient number of badges, they will be 'certified'. Badges were awarded, and tallied to gain certification as competency…”. The faculty members and web developer identified and scoped initial badge design from two in-person meetings, which the researcher then developed into the web software system. Concurrently, the instructional designer developed a separate badge design sourced from proposed instructional categories, which resulted in two conflicting gamification schemas. The full research team identified this schism during the implementation phase of initial instructional design into the web software. Analyses revealed three contradictions and mediations in the activity of developing and integrating the initial instructional activities into the web software. See Figure 8.

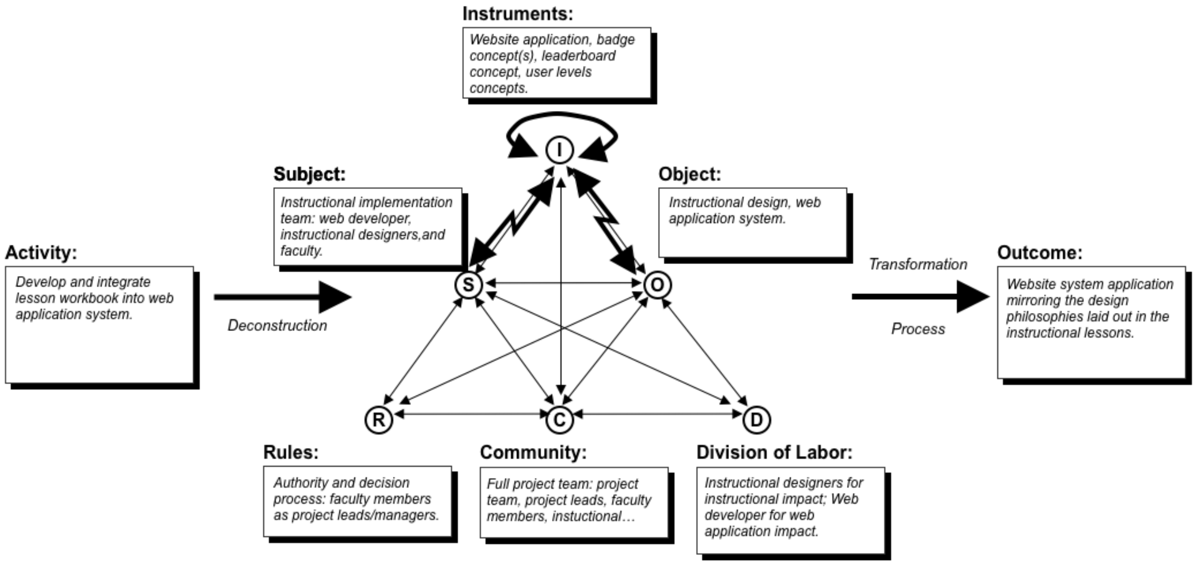

Instruments:
Website application, badge concept(s), leaderboard concept, user levels concepts.
Subject:
Instructional implementation team: web developer, instructional designers,and faculty.
Object:
Instructional design, web application system.
Activity:
Develop and integrate lesson workbook into web application system.
Deconstruction
Transformation
Process
Outcome:
Website system application mirroring the design philosophies laid out in the instructional lessons.
I
S
O
R
C
D
Rules:
Authority and decision process: faculty members as project leads/managers.
Community:
Full project team: project team, project leads, faculty members, instuctional…
Division of Labor:
Instructional designers for instructional impact; Web developer for web application impact.

Figure 8. Initial instructional design import activity system.

First, analysis uncovered a secondary resolved contradiction between the subject (web developer and instructional designer) and instrument (two conflicting gamification mental models). The web developer developed the web software following a gamification mental model derived from a December 2015 meeting that closely mirrored gamification implementations found in video games. In this model, a singular leaderboard collected and ranked user teams based on scores associated with claimed badges. Simultaneously, the instructional designer designed the instruction based on a separate, instruction-focused mental model of tightly-coupled leaderboards, badges, and levels. In this design, leaderboards were designed for each badge type and ranked by levels which could be achieved for completing particular goals. This contradiction was resolved through an additional round of instructional design by the instructional designer that paired the leaderboard concepts of instructional design to those outlined by the faculty members and web developer for the initial development cycle.

Second, analysis uncovered a primary resolved contradiction within instruments (badge/leaderboard mental models embedded within the web software and instructional design). Leaderboards were organized in the initial instructional concept by specific badge and levels per badge. Leaderboards were organized in the web software by a total sum of badge points per team, with no mention of levels. Badges were defined in the instructional content as singular objects representing a proficiency level based on points awarded. Badges were defined in the web software without levels and as a multiple-entity object, where one badge could virtually contain multiple variants containing similar attributes. This contradiction was resolved by centrally defining the leaderboard, badge, and user level concepts across the entire research team to a singular mental model of individual badge objects and a singular leaderboard.

Third, a secondary resolved contradiction was discovered between the instrument (gamification mental model from instructional design) and object (the web application implementation of instructional data formatting). This contradiction resulted in issues importing the instructional gamification placeholders into the web software’s gamification implementation. This was resolved through the congruence of instructional design gamification examples to match the design present in the web software, facilitating import of instructional lessons and gamification elements.

Gamification as ecommerce.

Late in development cycle 1, faculty members posited whether the web software could "give points for small accomplishments that can lead to badges, and thus give out fewer badges" and convert the gamification process into an ecommerce design. In the proposed ecommerce design, faculty members proposed a new in-software currency, experience coins, which would be redeemed to obtain badges. Analyses revealed two contradictions and mediations in the activity of incorporating experience coins into the web software as an ecommerce function to obtaining badges. See Figure 9.

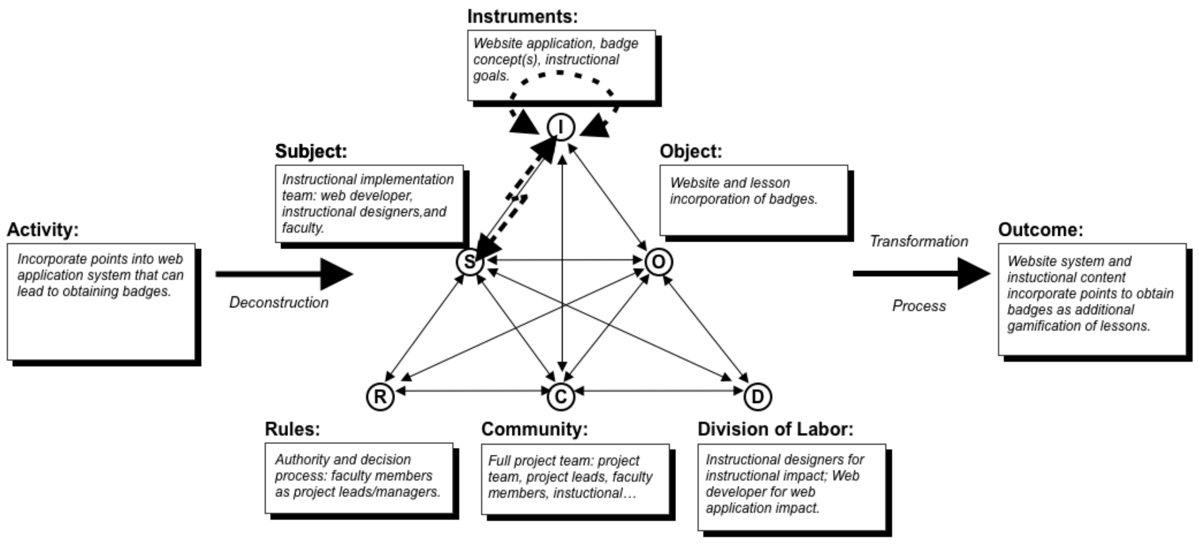

Instruments:
Website application, badge concept(s), instructional goals.
Subject:
Instructional implementation team: web developer, instructional designers,and faculty.
Object:
Website and lesson incorporation of badges.
Activity:
Incorporate points into web application system that can lead to obtaining badges.
Deconstruction
Transformation
Process
Outcome:
Website system and instuctional content incorporate points to obtain badges as additional gamification of lessons.
I
S
O
R
C
D
Rules:
Authority and decision process: faculty members as project leads/managers.
Community:
Full project team: project team, project leads, faculty members, instuctional…
Division of Labor:
Instructional designers for instructional impact; Web developer for web application impact.

Figure 9. Badge/point gamification decoupling activity system.

First, analysis uncovered a secondary unresolved contradiction between subject (faculty members and web developer) and instrument (the proposed ecommerce mental model and the existing badge mental model). The faculty member ecommerce proposal decoupled points from badges into experience coins, which conceptualized both entities as independent data objects where experience coins could be submitted for badges. The existing web developer mental model of gamification configured points as a badge component, which was used to drive further web development. No mediation was made on how experience coins would interact with the existing badge designs.

Second, analysis uncovered a primary unresolved contradiction within instruments (the proposed experience coin design mental model and the existing web software gamification data schema). Neither instrument concept was compatible with the other. No mediation was made on how the web software badge model would be reconfigured to meet the activity transformation process.

The two unresolved contradictions halted the activity process. The identified unresolved contradictions in the previous activity system were resurrected in development cycle 2 as a gamification redesign. During the development process of cycle 2, faculty and instructional designer team members proposed manual and automatic assessment scoring to the web developer, who would interpret the proposals from a software development perspective and respond with additional probing questions to situate the proposed changes within the web software. Initial email discussions between faculty members, the instructional designer, and the web developer resulted in the web developer drafting an updated gamification design which included a placeholder for experience coins that could be submitted for badges. See Figure 10. The drafted gamification design restarted the ecommerce experience coin proposal activity.

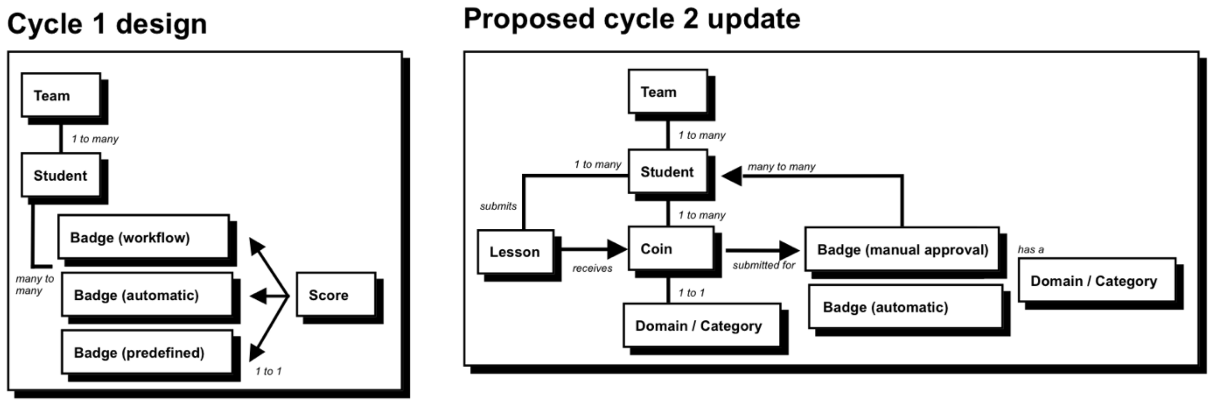


Figure 10. Cycle 2 ecommerce gamification redesign data changes.

Analyses revealed two contradictions and mediations in the activity of incorporating updated gamification elements (coins, badges, levels) into the web software. See Figure 11.

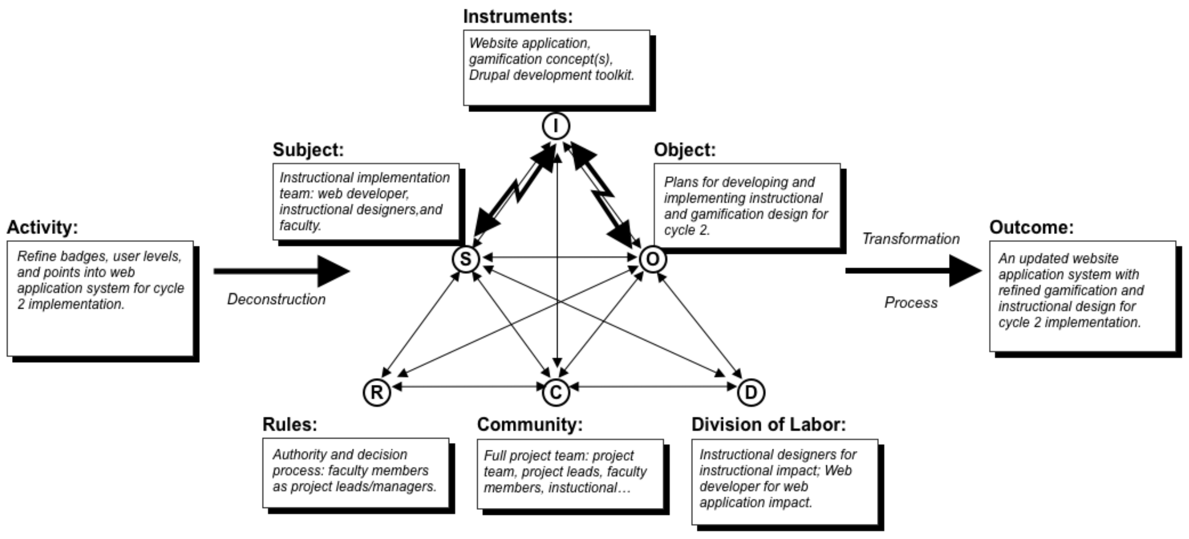

Instruments:
Website application, gamification concept(s), Drupal development toolkit.
I
Subject:
Instructional implementation team: web developer, instructional designers,and faculty.
Object:
Plans for developing and implementing instructional and gamification design for cycle 2.
Activity:
Refine badges, user levels, and points into web application system for cycle 2 implementation.
Deconstruction
S
O
Transformation
Process
Outcome:
An updated website application system with refined gamification and instructional design for cycle 2 implementation.
R
C
D
Rules:
Authority and decision process: faculty members as project leads/managers.
Community:
Full project team: project team, project leads, faculty members, instuctional…
Division of Labor:
Instructional designers for instructional impact; Web developer for web application impact.

Figure 11. Ecommerce instructional design proposal activity system.

First, analysis uncovered a primary resolved contradiction between subject (web developer and faculty/instructional design members) and instrument (existing gamification mental model, and the ecommerce experience coin mental model). This tension resulted in an unclear understanding from team members on how points and experience coins were incorporated with badges. The web developer was unclear on the process of redeeming experience coins for badges, using their pre-established mental model as a foundation. The instructional designers and faculty resurrected the mental model of experience coins that could be redeemed for digital badges. This tension was mediated by the faculty members agreeing to a simplified ecommerce mental model meshing both perspectives for the web software. From this point on, badges were reframed as experience coins that could be redeemed for real-world prizes throughout the project.

Second, analysis uncovered a secondary resolved contradiction between instrument (experience coin mental model) and object (ecommerce instructional design implementation for manually-assessed coins). The pre-existing instructional design implementation paired assessment completion to receive experience coins. The instructional design object had not standardized a pairing process of immediate rewarding mental model for both automatically assessed and manually assessed experience coins. In the proposed instructional design model, all experience coins would be immediately rewarded whether the assessment was manually or automatically scorable. For manually-scorable experience coins, an automatically awarded experience coin value could change after an administrative manual review. This tension was mediated by a project team decision to move away from the proposal to automatically give all experience coins immediate feedback and obtainment. Instead, the research team relied on the existing web application system which separated automatic and manual feedback assessments. Automatic assessment-configured experience coins would be automatically redeemed at assessment completion. However, experience coins paired with manual assessment would be individually redeemed by a team member. See Figure 12 for the mediated experience coin design. This mediated design and its scaffolded consequences prompted an additional contradiction activity, manual assessment review, analyzed further below.

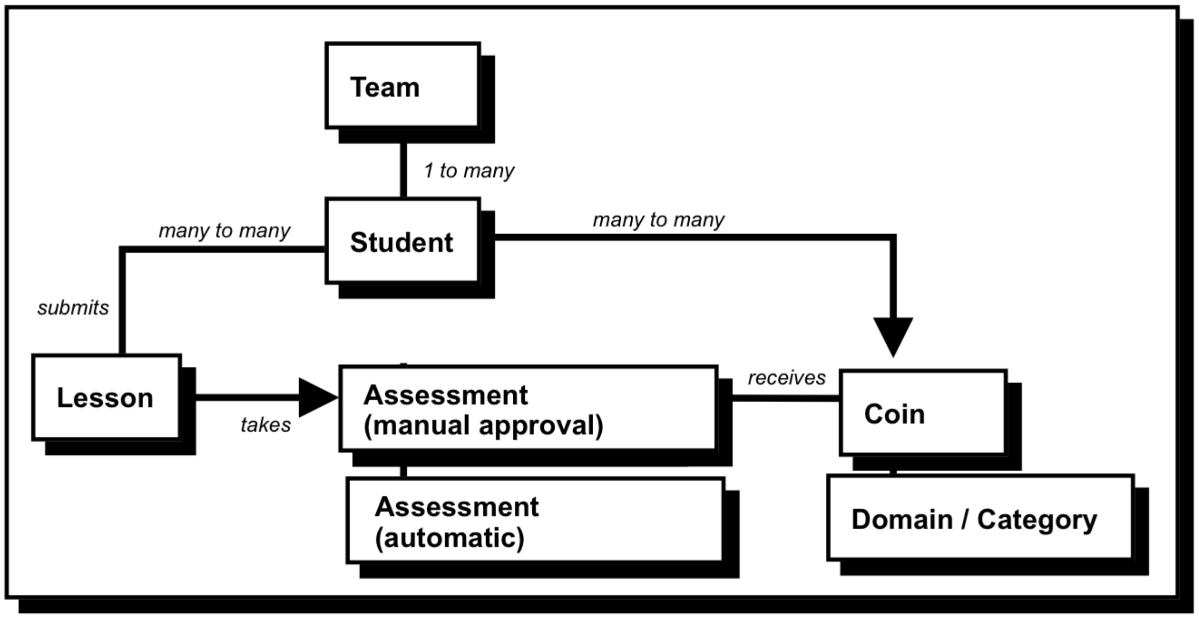


Figure 12. Cycle 2 ecommerce gamification redesign data changes.

Badges and team profiles.

Faculty members proposed an implementation of the refined gamification elements outlined above into the web software’s student team profiles during cycle 2 development. The original profile design did not showcase any experience coin information for team members. The proposal updated team profiles to show experience coin counts within each domain category for each team member. The researcher developed profile design prototypes, refined through multiple rounds of research team feedback, seen in Figure 13. Subsections 1 and 2 were team profiles after cycle 1 development. Subsection 3 was a prototyped user profile before the ‘gamification as ecommerce’ activity above. Subsections 4 and 5 were mediated low-fidelity design prototypes developed by the web developer within the activity, which were converted to increased fidelity in subsections 6 and 7.

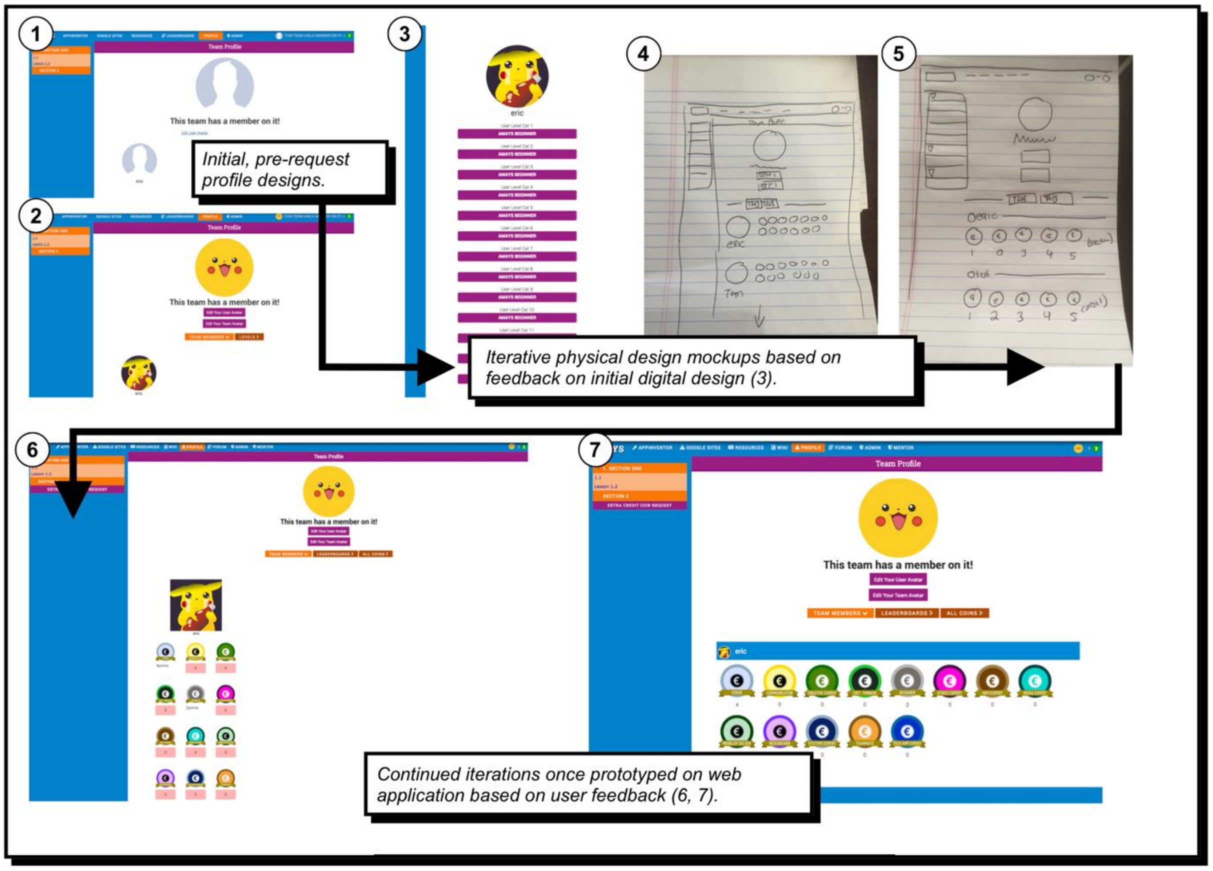

1
2
3
4
5
6
7
Initial, pre-request profile designs.
Iterative physical design mockups based on feedback on initial digital design (3).
Continued iterations once prototyped on web application based on user feedback (6, 7).
Team Profile
This team has a member on it!
Edit Your User Avatar
Edit Your Team Avatar
TEAM MEMBERS
LEADERBOARDS
ALL COINS
eric

Figure 13. Team profile redesign process.

Analyses revealed two contradictions and mediations in the activity of re-designing team profiles within the web software. See Figure 14.

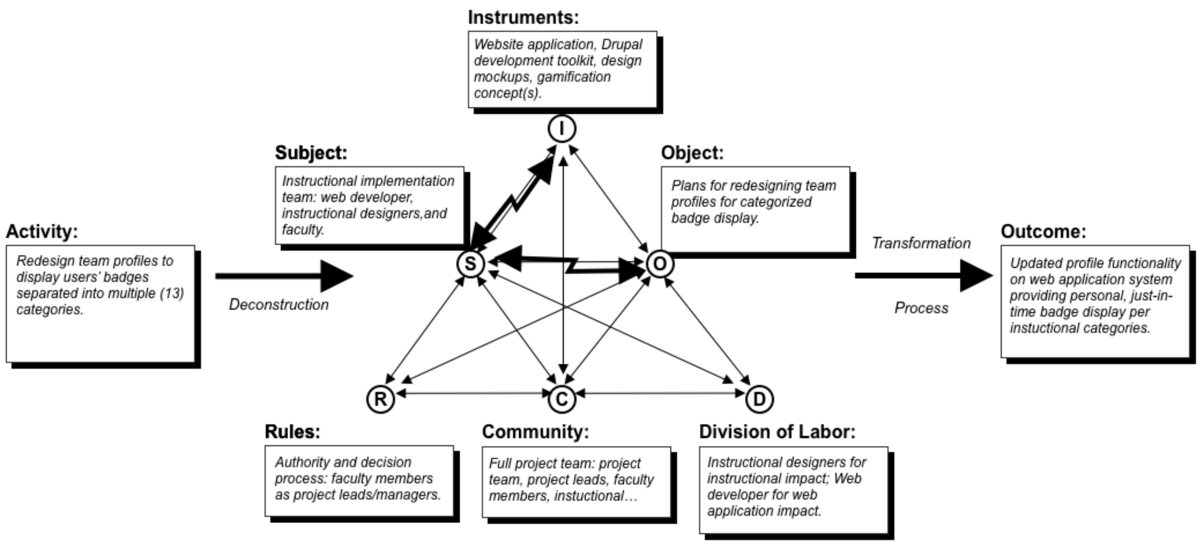

Instruments:
Website application, Drupal development toolkit, design mockups, gamification concept(s).
I
Subject:
Instructional implementation team: web developer, instructional designers,and faculty.
Object:
Plans for redesigning team profiles for categorized badge display.
Activity:
Redesign team profiles to display users' badges separated into multiple (13) categories.
Deconstruction
S
O
Transformation
Process
Outcome:
Updated profile functionality on web application system providing personal, just-in-time badge display per instuctional categories.
R
C
D
Rules:
Authority and decision process: faculty members as project leads/managers.
Community:
Full project team: project team, project leads, faculty members, instuctional…
Division of Labor:
Instructional designers for instructional impact; Web developer for web application impact.

Figure 14. Team profile redesign activity system.

First, analysis uncovered a secondary resolved contradiction between subject (web developer and faculty members) and instrument (design mockups and design mental models). Design mockups shifted between multiple web developer-created prototypes through faculty feedback, as detailed above. This contradiction was mediated through iterations of analysis, feedback, and refinement, which blended contradicting mental models into a design mockup which met project needs for both subjects.

Second, analysis uncovered a secondary resolved contradiction between subject (web developer and faculty/instructional designers) and object (technical profile implementation and instructional implementation). The instructional implementation proposed by faculty and instructional design members included customized, per-coin ribbon graphics of increasing ‘quality’ (bronze, silver, and gold) to designate experience coin proficiency in a domain area. The web developer had cautioned that this request would require three additional graphics variants for each of the 13 coin types. This contradiction was mediated through repurposing already-made experience coin graphics rather than additionally creating new ribbon graphics, placing a counter below each experience coin.

Manual assessment review.

As described above, cycle 2 implemented open-ended, manually-scored experience coins that were not automatically assessed by the web software. The web developer created a manual experience coin assessment process and administrative page for team members to approve or decline manually-scored experience coins. After piloting the design in cycle 2, cycle 3 was focused on a broader, longer implementation of the instructional design and web software. Analyses revealed two contradictions in the activity of developing and deploying manual experience coin assessment for cycle 3 implementation. See Figure 15.

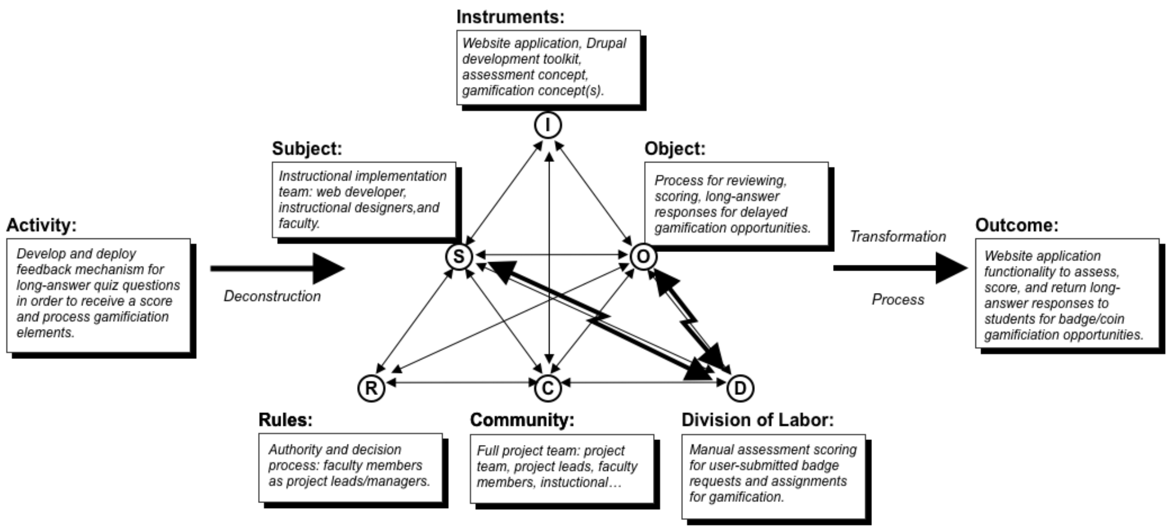

Instruments:
Website application, Drupal development toolkit, assessment concept, gamification concept(s).
I
Subject:
Instructional implementation team: web developer, instructional designers,and faculty.
Object:
Process for reviewing, scoring, long-answer responses for delayed gamification opportunities.
Activity:
Develop and deploy feedback mechanism for long-answer quiz questions in order to receive a score and process gamificiation elements.
Deconstruction
S
O
Transformation
Process
Outcome:
Website application functionality to assess, score, and return long-answer responses to students for badge/coin gamificiation opportunities.
R
C
D
Rules:
Authority and decision process: faculty members as project leads/managers.
Community:
Full project team: project team, project leads, faculty members, instuctional…
Division of Labor:
Manual assessment scoring for user-submitted badge requests and assignments for gamification.

Figure 15. Manual experience coin assessment activity systems.

First, analysis uncovered a resolved secondary contradiction between subject (instructional designer and web developer) and division of labor (manual assessment scoring delegation). The web developer constructed an administrative page for reviewing and scoring manual experience coin assessments. The web developer did not delegate any team member role as the point-of-contact for scoring the requests, implicitly expecting the instructional designer to fulfill this role. However, the instructional designer was not familiar with how to locate and confirm manually-scored experience coin requests. This contradiction was resolved through explicit definitions of division of labor between subjects and defining manual experience coin request expectations for each subject role. The instructional designer was delegated to review and approve manual experience coin requests. The web developer had no formal role definition.

Second, analysis uncovered a secondary resolved contradiction between division of labor (manual coin process delegation) and object (the website implementation of manual coins). The initial cycle 2 design of manual experience coins focused on passive checks of the web software to locate new manual request submissions. Building on the mediation from the first contradiction, the delegated instructional designer proposed a process change to have the web software actively notify administrative users upon new manual experience coin requests. This mediation was resolved through the creation of an email notification process within the transformation process by the web developer.

Results

In the current study, the researcher identified a series of activity system tensions that enhanced these difficulties during software development for a large IDT project. Analysis of identified contradictions and mediations revealed three main sources of tension within project activities during three cycles of development: Division of Labor, Knowledge Transfer, and Mental Model Discrepancies.

Project Team Division of Labor

Division of labor activity negotiations uncovered difficulties with software-mediated administrative processes and tasks which could not be automated. Figure 14 highlighted the contradictions present in the manual experience coin approval process, which required scaffolded notifications and emails to alert team members of new submissions. This analysis supported findings of Domínguez et al. (2013) that more-immediate feedback for gamification must not only occur for learners but also for instructors and facilitators. While automation may be alluring to minimize this problem, technology has not yet perfected automated, open-answer assessment review. At the time of writing, assessment of ill-structured problems with no single correct solution required scaffolds, such as rubrics or criterion, to construct assessment and feedback – and most software packages currently facilitating this level of assessment are constructed for a particular flavor or subject area of activity feedback (von Wangenheim et al., 2018). Mirroring the findings of Douce, Livingstone, and Orwell (2005), software systems should aim to support, not replace, human assessment. IDT software design and development processes should approach considerations to minimizing the operational burdens of ill-structured assessment.

As an example, a novel and inexpensive solution could, rather than scaffolding an automatic assessment technology, leverage the existing software application to reduce the manual assessment process. This perspective would position its activity around the goal of identifying manual assessment needs and redirecting the appropriate division of labor to complete the task as efficiently as possible. Process reductions could include responsive notification creation and distribution, visually-distinct administrative quick links, in-application notification messages, or role-based processes or workflows. This perspective would minimize additional technical debt burdens in order to implement or design an automatic assessment platform. Additionally, this perspective would refocus effort on the primary, underlying need – a prompt turn-around and well-defined workflow between assessment submission and feedback.

Project Team Knowledge Transfer

Toader and Kessler (2018) posited that teams faced difficulties when trying to transfer knowledge between ill-structured problems and tasks. Activity negotiations presented in this chapter strengthened that argument, identifying contradictions in team knowledge transfer between subject roles. Knowledge transfer contradictions were solely constrained between the intersection of IDT and computer science: instructional designer/web developer (see Figures 8, 11, and 15) and faculty/web developer (see Figures 9, 11, and 14). This finding echoed the need for greater cross-pollination between the two fields, as proposed by Adnan and Ritzhaupt (2018). Additionally, knowledge transfer paired to mental model discrepancies, where mediations facilitated the transfer of knowledge throughout project team roles. Secondary contradictions between subject roles and instruments were the most common contradiction observed (see Figure 7).

This commonality does not suggest that knowledge transfer was absent between roles. Rather, it demonstrated the difficulties of knowledge transfer accessibility, as investigated by Persky and Murphy (2019), where transferred knowledge could be reactivated in new contexts as a product of fluency. DBR provided opportunities to situate knowledge transfer and fluency as lensed from project phases. Within this study, analysis uncovered the presence of mediated knowledge transfer delay which resulted from the project phase within a DBR cycle, as seen in Figures 9 (unresolved) and 11 (resumed). The activity transformation was determined by the project team to better situate within DBR cycle phases which were appropriate for a successful mediation process. This scenario highlighted the rigid implementations of broadly-scoped DBR project phases, where testing phases (as seen in Figure 9) were not positioned for large-scale software changes compared to planning phases (as seen in Figure 11). This scenario extended the findings of Getenet (2019), who strengthened DBR's role to situate knowledge transfer between project team members, by positioning the accessibility of knowledge transfer within DBR cycle phases.

Mental Model Discrepancies

Activity negotiations presented in this chapter uncovered difficulties in consolidating conceptual understandings between instructional goals and software implementation, as seen in Figures 8, 9, and 11. Mental models constructed to design the underlying software proved inflexible after their translation into reality. Badging software design mental models failed to quickly adapt to instructional models that invoked flexible and interchangeable conceptual models, ideas, and proposals in order to satisfy better-defined instructional goals or outcomes. Identified contradictions were in part resolved by delaying major changes in mental models until future development cycles, bolstering evidence for the time-consuming nature of software development (Sanders et al., 2014). This delay leveraged iterative development concepts, found within rapid prototyping and design-based research (The Design-Based Research Collective, 2003; Tripp & Bichelmeyer, 1990), to explore (see Figure 9) or implement (see Figure 11) feedback-prompted software changes during subsequent planning and development cycles. This should be expected, as previous research found instructional analysis of a designed system in authentic contexts resulted in successive refinements and analysis (Kali & Ronen-Fuhrmann, 2011). However, these considerations approached contradictions in a reactive context, in that identification of these incongruences occurred at critical moments of implementation (see Figure 8) or wind-down (see Figure 9). Both scenarios outlined project phases where retroactive analysis would realistically begin to take shape, which limited mediation opportunities. Affordances should be made to proactively identify and facilitate incongruences throughout all phases of design and development process, as its effects would scaffold throughout the project.

Analyses uncovered multiple shifts of the underlying gamification design in an effort to mirror continually-refined instructional goals, seen in Figures 8, 9, and 11. To alleviate this reactive context, project team members could process instructional software requests through dual-impact design considerations: (1) impact to the developed software component, and (2) impact on the broader intersection of software and instructional design. This multi-faceted instrument analysis of developed IDT research software would situate technology implementations alongside instructional design for analysis (Oliver, 2011) and unmask the software development process (Shavelson et al., 2005), facilitating opportunities for software replication or re-use. This perspective echoed the premise of instructional development research explored by Richey (1997), in that instructional software approaches should position the software in a similar dual-impact consideration: (1) a multi-faceted instrument housing both the physical instrumentation (software itself), as well as (2) the conceptual models making up the instructional design (software data structures of instructional design). Both instructional and software development should take proactive care to ensure both elements remain in-sync throughout the project.

Discussion

RQ1. How did social contradictions and mediations shape the software development process within a large educational research project?

Social contradictions and mediations shaped the software development process through two primary design considerations: instructional/technological perspective negotiations and technology-mediated auxiliary support function design. This section applied each consideration as design principles to guide broad-scoped application into future IDT software development projects and more-broadly distribute best-practices (The Design-Based Research Collective, 2003). Design principles followed the formatting of T. Park and Lim (2019).

Instructional/technological perspective negotiations.

The negotiations between instructional- and technological-focused perspectives, as seen in project team knowledge transfer and mental model discrepancies, revealed issues in knowledge and mental model transfer. Echoing the findings by Adnan and Ritzhaupt (2018), discussions between team members about non-functional requirements (NFRs) were crucial to the quality of the software development process. NFRs, the understandings of a software's quality attributes, provide blueprints for project team members to apply within their perspective domains, which bound the project's scope for software design and development (Mahmoud & Williams, 2016). NFRs act much like user journeys in web development, providing broad, non-technical explanations of what the designed software should operate as, such as "student team members should be able to access their team profile and view their badges, scores, and ranking for every member". Findings revealed schisms in the reporting of NFRs as evidenced by consequential mental model and knowledge transfer gaps, as team member roles attempted to self-construct their own NFRs. Those self-constructed and self-contained NFRs, seen from CHAT as a mental model instrument, clashed when paired with another self-constructed NFR on the same activity.

Findings revealed that knowledge transfer of individualized NFRs were positioned reactively the presence of contradictions. This positioning situated contradictions not as a mediative purpose, but purposed as knowledge distribution which facilitated mediation. There are multiple pathways to alleviate this reactive perspective. One option would be to call on project leads – usually principal investigators – to facilitate the creation of NFRs. However, this option does not facilitate the presence of emergent NFRs throughout the project and does not situate its continual communication to all team members throughout the project. Rather than tasking project leads to specifically that purpose, the ‘continual communication’ concept can be implemented from a project-wide standpoint. Project management tools should shift to better situate active communication strategies for either constructing or identifying NFRs early and throughout the project. DBR positioned itself as a research methodology to foster collaboration between disciplines, but failed to situate the pairing of knowledge transfer and effective collaboration. This active communication would mitigate effects of a dysfunctional organizational climate or a recently-established team interaction (Nielsen & Cappelen, 2014), and cut through the knowledge transfer barriers of cultures of the individual, team, organization, and inter-organization (P. Y.-T. Sun & Scott, 2005). Placement of this continual communications strategy within the project management would alleviate the task of soliciting and communicating NFRs from and to individual members. Instead, the placement would embed itself both as a central tenant throughout each member's role in the project, and within the project management itself, pre-mediating emergent cultural contradictions.

> **Principle of Continual Communications: Provide continual, proactive communications - situated within project management – to centrally facilitate and assess software non-functional requirements and knowledge sharing throughout the team.**

Technology-mediated auxiliary support function design.

The presence of cycle 3 contradictions regarding manual assessment review revealed issues with technology-mediated auxiliary function's placement within the IDT software project. Findings revealed one identified auxiliary support function design – manual assessment scoring – and the mediating creation of active notifications to replace passive status checks. The presence of a contradiction from support subject roles expands NFR's lens beyond primary learner roles to include auxiliary roles and functions. The auxiliary function roles are intended to minimize operational burdens supporting the software's instruction, rather than piling technological debt on top of the realized-software development to accommodate automation. The presence of these auxiliary functions forces project considerations of downstream support issues introduced by the creation of interventions and components. Considerations should be made from two perspectives as found by Kultur, Oytun, Cagiltay, Ozden, and Kucuk (2004): new roles and responsibilities, and changes in work practices.

New roles and responsibilities should be actively communicated to both the applicable subjects and the project team. Applicable subjects should have a clear awareness of the expected duties associated with new roles and responsibilities. This awareness provides an opportunity for the applicable subjects to also shape the intended support structures to match their historical and cultural experiences, as seen within Manual Assessment Review. Additionally, the communication to the project team can facilitate knowledge transfer of the auxiliary support process throughout the team to better situate any future development cycles, feature requests, or support role backups.

Change in work practices should be situated to minimize passive functions and maximize on-demand active functions. As seen in the Manual Assessment Review, passive auxiliary support functions presented two problems. First, the passive component extended auxiliary function turnaround time by tying a request to the most recent role check-in to the system. The system should, whenever possible, instead approach active scaffolds to provide notifications, alerts, or messages to applicable subjects in order to reduce the turnaround time. Second, the passive component tacked on unnecessary time spent on checks, which is doubly impactful if the applicable subjects are also embedded in additional project roles. The unnecessary time allotment also presents concerns for building support expertise if process repetition is stifled by unsuccessful passive checks. Auxiliary support mechanisms should focus administrative actions on quick turnaround time to reduce operational burden.

- **Principle of Mediated Auxiliary Support: Design and develop educational software administrative and support functions, paired to instructional design and development, in order to minimize burdens of new roles/responsibilities and in changes to work practices.**

Iterative & Integrative Instructional Software Design (IIISD)

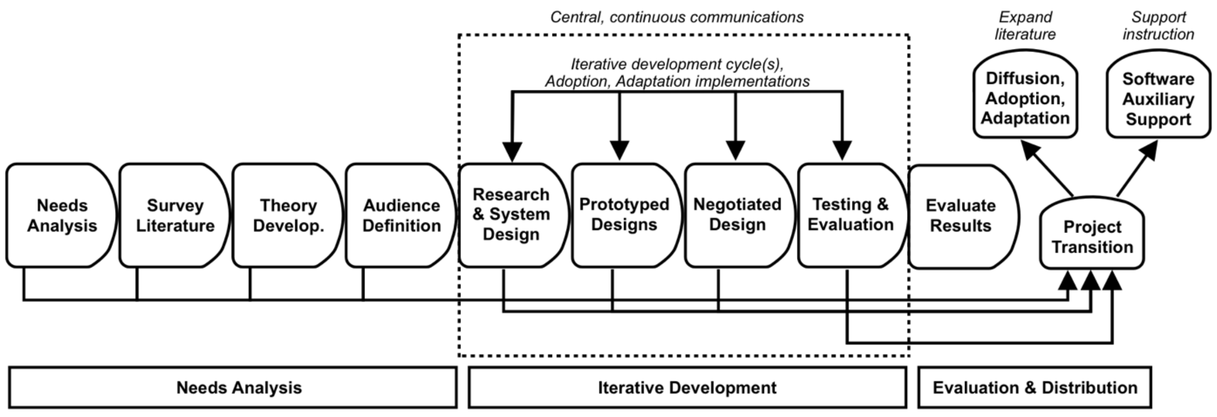


Figure 16. Iterative & Integrative Instructional Software Design framework.

To mitigate contradictions found within the software development from the perspectives of division of labor, knowledge transfer, and mental model discrepancies, the researcher proposes a revised software development framework as seen in Figure 15 - the Iterative & Integrative Instructional Software Design framework (IIISD) - to accommodate and direct knowledge communication throughout the development process. The IIISD was founded in the integrative work of Bannan-Ritland (2003) and the Integrative Learning Design (ILD) framework and the stymied progress of software reuse in IDT research uncovered by the researcher in Chapter 1. Additionally, IIISD mediated socially-aware context to consciously and systematically distribute project, task, and domain knowledge through continuous team communication.

IIISD redefined re-use previously defined by ILD into a modern-day software development context. ILD defined diffusion as the distribution of code or design to a wider audience for re-use and replication. ILD situated adaptation and adoption involving the consumption of outside code or design within a project – adaptation required customizations for usage, while adoption was intended for more plug-and-play context. As realized in Sembrat (2019), software re-use could take the form of either small components or large-scale software projects, so the terms of adaptation and adoption were mutated slightly to accommodate this broader scope. In order to accommodate the modern software development best-practices, IIISD separated internal re-use from external re-use, as familiarity with the written code differed greatly between the two scopes. In internal re-use, the code had historical, cultural, and social context by which the intent and meaning is more easily understood. In essence, internal re-use would be implemented simply as an additional development iteration. External re-use, however, was the embedded action of adaptation or adoption from outside the project scope into a new and unfamiliar environment. There would be no expected code familiarity, resulting in code that would be written, commented, designed, and understood from a different social context, echoing Panke et al. (2007). To accommodate this, re-use is re-situated to be imported throughout the development process to satisfy the limitless opportunities in the development process to re-use software or design. Additionally, these changes could impact succeeding or proceeding steps, requiring a fluid and interconnected development process.

Adnan and Ritzhaupt (2018) identified agile project management as an opportunity for educational research to learn from software engineering. Specifically, the researchers expressed value in the agile practices of recurring and frequent meetings (scrum meetings), periodic development deliverable timelines (sprints), and continual knowledge transfer and collaboration across team members. This knowledge transfer was defined as ‘cross-functionality through cross-fertilization’ (Barke & Prechelt, 2018). Rather than requiring each project adhere to strict agile methodological management philosophy, iterative design philosophies such as design-based research (DBR) provided scaffolds by which to embed communication strategies. IIISD’s central and continuous communications focused on three communication pillars with the expressed intent of knowledge sharing: 1) periodic, strictly-defined, recurring meetings following a quick agile methodology, 2) short-term development iteration cycles for review and refinement, and 3) centralized communications strategies. Each pillar applies the discussion from Instructional/Technological Perspective Negotiations above.

Periodic, strictly-defined recurring meetings would provide each team member a project status update and the ability to identify project-blockers. While this aspect is not intended to strictly adhere to the daily scrum standup as detailed in Sonya Zhang and Dorn (2012), the general format emulated the scrum design. Each meeting member would provide a three-pronged update: 1) “what have you been working on?”, 2) “what do you plan on working on next?”, and 3) “what issues or problems have you encountered that prevent you from completing your tasks?”. These queries intend to promote short status-updates which can foster additional communications to rectify issues. Additionally, the meeting would facilitate the distribution of task and project knowledge across all meeting attendees, affording each member a higher-level project understanding and the tasks and projects of their peer team members. Finally, these meetings would situate and provide a progress update for the iterative development cycles from each contributor.

Short-term development iteration cycles mirrored the combinations of micro-cycles into a meso-cycle in DBR (Pool & Laubscher, 2016), in that simultaneous aspects of a project are intertwined in the process of exploration-construction-evaluation towards a completed iteration (Parmaxi & Zaphiris, 2015). A project lead would determine the length and goals of each iteration cycle, with the understanding that shorter-term iterations would allow for movement, exploration, and analysis of new ideas without the full invested resources of carrying a prototyped design to fruition. This shorter-termed timeline also would encourage a design philosophy of prototyped designs interacting with activity components in its transformation into a mediated, negotiated design decision – itself an activity system in action. This design philosophy would focus review and feedback not on completely integrated and realistic prototypes, but multiple prototypes of increasing fidelity and realism as they process through approvals and feedback towards a negotiated design. This would afford a development strategy primed to identify issues and potential blockers as the prototype evolves into greater fidelity.

Finally, the usage of central communication strategies would accomplish two main goals. First, the usage of observable and recommended communication channels would facilitate transparency and knowledge-building for team members to build competency and expertise in an area. This idea would better facilitate the distribution of knowledge from expertise member(s) across the entire team. Second, the usage would provide defined outlets for team members to ask questions, solicit feedback, or showcase a deliverable to the project team. This central, continuous communication strategy would encourage and facilitate collaboration while also preserving knowledge for reference or additional (or new) team members. No specific tool is purposefully identified to accommodate adapting communication strategies to team size, preferences, available tools, and timelines. This process would only require that a team member serve as the communication lead for recurring meetings and directing discussion-oriented communications, a role that can naturally coincide with a project lead.

IIISD positions project evaluation and distribution from two transitional perspectives. First, the project transition approaches re-use in order to expand the literature. As stated above, re-use was redefined to match the broader interpretations defined in Chapter 1’s second analysis phase. These updated definitions focus extensions of the literature either through explicit code diffusion, adoption, or adaptation, or a broader, academic examination of the software (as commonly seen in IDT research, per Chapter 1). Second, the project transition focuses on the developed software as it pertains to the auxiliary support, as described in Technology-mediated Auxiliary Support Function Design above. IIISD does not explicitly position the application of the developed software to deliver instruction to learners within the framework, applying the argument of Oliver (2011) that instructional software should be positioned as an instrument worthy of its own examination. However, this does not discount that instruction is an expected outcome of developed software within IDT. As such, the framework needs to accommodate the software development life-cycle for the ongoing auxiliary services in order to reduce technical debt and maximize instructional value for learners within its production (authentic) environment. This component pairs with the production phases mapped in the evolution of the software development life cycle (Kneuper, 2017), which identified the post-development production phases throughout 60 years of software development.

Conclusion

This case study analyzed the social influences and contradictions of gamification implementation situated within an IDT software development project through CHAT. Findings revealed social contradictions of mental model, knowledge transfer, and labor division negotiations, each centered around the translation of instructional goals and requirements to software implementations and designs. The identification and analysis of social contradictions prompted the creation of two design principles to better communicate software development best-practices.

These design principles were applied in the creation of a design framework, IIISD, to better situate socially-conscious project team activities into software development. IIISD is designed to better represent the current software development landscape within IDT research and facilitate the social negotiations impacting and influencing software development. Additionally, IIISD is positioned as a response to more than decade of software development studies in IDT research as analyzed by Sembrat (2019) and as an evolution of ILD to modern-day software development practices to better position the software development process from social lenses. IIISD's purposeful abstraction is intended to provide flexibility with both IDT methodologies and successful software development best-practices outside of the field.

Limitations and Considerations

This main limitation of this study is that its CHAT analyses is positioned from a singular web developer role of the primary author. Future IDT studies could conduct CHAT analyses from software developers within other IDT software case studies would be valuable to validate or refute the method. Additional analyses would produce additional or supportive design principles, in order to extend the literature through positioning validated findings into design patterns. Construction of synthesized design patterns from aggregated design principles, as suggested by The Design-Based Research Collective (2003), could help to more-broadly distribute best-practices at the intersection of IDT and computer science. Future IDT studies using IC for data coding would validate or refute its analytical framework, given its limited usage outside of health sciences. Additionally, additional studies applying the IC deviations outlined in the data analysis would be valuable to validate or refute the adjusted methodology and the chronological analysis phase for qualitative project data.

The researcher hopes additional studies will incorporate the IIISD framework into authentic contexts to understand its situation of IDT software development projects. IIISD is not intended as an end-product model – rather, it should evolve through successive adoption and adaptation to validate or refute its findings as a process towards construction of design principles and best-practices. Considerations should be taken into how to properly revision a framework to match environmental evolutions, noting ambiguity in generational activity theory terminology as shown in Sannino and Engeström (2018). Similar to the evolution of ILD into IIISD, consideration should be taken within future studies to ensure that IIISD is compatible with future-state software development and project management best-practices, as both areas are evolving alongside technological advances outside the field of IDT research. From historical observation of older software studies as seen in Sembrat (2019), these advances would influence development tools and platforms utilized within the study.

References

Adnan, N. H., & Ritzhaupt, A. D. (2018). Software engineering design principles applied to instructional design: what can we learn from our sister discipline? *TechTrends: Linking Research and Practice to Improve Learning, 62*(1), 77-94.

Amiel, T., & Reeves, T. (2008). Design-based research and educational technology: rethinking technology and the research agenda. *Journal of Educational Technology & Society, 11*(4), 29-40.

Bannan-Ritland, B. (2003). The role of design in research: the integrative learning design framework. *Educational Researcher, 32*(1), 21-24. doi:10.3102/0013189x032001021

Barab, S. A., Barnett, M., Yamagata-Lynch, L., Squire, K., & Keating, T. (2002). Using activity theory to understand the systemic tensions characterizing a technology-rich introductory astronomy course. *Mind, Culture & Activity, 9*(2), 76-107. doi:10.1207/S15327884MCA0902_02

Barke, H., & Prechelt, L. (2018, 27 May-3 June 2018). *Some reasons why actual cross-fertilization in cross-functional agile teams is difficult.* Paper presented at the 2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE).

Barnett-Page, E., & Thomas, J. (2009). Methods for the synthesis of qualitative research: a critical review. *BMC Medical Research Methodology, 9*(1), 59. doi:10.1186/1471-2288-9-59

Barron, B. (2004). Learning ecologies for technological fluency: gender and experience differences. *Journal of Educational Computing Research, 31*(1), 1-36. doi:10.2190/1N20-VV12-4RB5-33VA

Barroso, J., Gollop, C. J., Sandelowski, M., Meynell, J., Pearce, P. F., & Collins, L. J. (2003). The challenges of searching for and retrieving qualitative studies. *Western Journal of Nursing Research, 25*(2), 153-178. doi:10.1177/0193945902250034

Birch, P. D. J., & Whitehead, A. E. (2020). Investigating the comparative suitability of traditional and task-specific think aloud training. *Perceptual & Motor Skills, 127*(1), 202-224.

Borro-Escribano, B., Del Blanco, Á., Torrente, J., Alpuente, I. M., & Fernández-Manjón, B. (2014). Developing game-like simulations to formalize tacit procedural knowledge: the ONT experience. *Educational Technology Research and Development, 62*(2), 227-243. doi:10.1007/s11423-013-9321-6

Boyd, D. W. (1993). The new microcomputer development technology: implications for the economics instructor and software author. *The Journal of Economic Education, 24*(2), 113-125. doi:10.1080/00220485.1993.10844785

Branon, R., Wolfenstein, M., & Weiss, C. (2016). MASLO: a mobile learning development system. *International Journal of Designs for Learning, 7*(3). doi:10.14434/ijdl.v7i3.13116

Çakıroğlu, Ü., Kokoç, M., Kol, E., & Turan, E. (2016). Exploring teaching programming online through web conferencing system: the lens of activity theory. *Journal of Educational Technology & Society, 19*(4), 126-139.

Carroll, J. M., & Olson, J. R. (1987). *Mental models in human-computer interaction: research issues about what the user of software knows*: National Academy Press.

Carvalho, M. B., Bellotti, F., Berta, R., De Gloria, A., Sedano, C. I., Hauge, J. B., . . . Rauterberg, M. (2015). An activity theory-based model for serious games analysis and conceptual design. *Computers & Education, 87*, 166-181. doi:10.1016/j.compedu.2015.03.023

Chen, M.-P., Wong, Y.-T., & Wang, L.-C. (2014). Effects of type of exploratory strategy and prior knowledge on middle school students' learning of chemical formulas from a 3D role-playing game. *Educational Technology Research and Development, 62*(2), 163-185. doi:10.1007/s11423-013-9324-3

Choi, H., & Kang, M. (2010). Applying an activity system to online collaborative group work analysis. *British Journal of Educational Technology, 41*(5), 776-795. doi:10.1111/j.1467-8535.2009.00978.x

Clark, R. E. (1983). Reconsidering research on learning from media. *Review of Educational Research, 53*(4), 445-459. doi:10.2307/1170217

Cobb, P., Confrey, J., diSessa, A., Lehrer, R., & Schauble, L. (2003). Design experiments in educational research. *Educational Researcher, 32*(1), 9-13. doi:10.3102/0013189X032001009

Collis, B., & Margaryan, A. (2004). Applying activity theory to computer-supported collaborative learning and work-based activities in corporate settings. *Educational Technology Research and Development, 52*(4), 38-52. doi:10.1007/BF02504717

Computers & Education. *Computers & education journal*. Retrieved from https://www.journals.elsevier.com/computers-and-education/

Cordero, K., Nussbaum, M., Ibaseta, V., Otaíza, M. J., Gleisner, S., González, S., . . . Carland, C. (2015). Read create share (RCS): a new digital tool for interactive reading and writing. *Computers & Education, 82*, 486-496. doi:10.1016/j.compedu.2014.12.006

Dagdilelis, V., Evangelidis, G., Satratzemi, M., Efopoulos, V., & Zagouras, C. (2003). DELYS: a novel microworld-based educational software for teaching computer science subjects. *Computers & Education, 40*(4), 307-325. doi:10.1016/S0360-1315(02)00133-1

David, H., & Victor, T. (2002). Learning within the context of communities of practices: a re-conceptualization of tools, rules and roles of the activity system. *Educational Media International, 39*(3-4), 247-255. doi:10.1080/09523980210166468

De Boeck, P., & Minjeong, J. (2018). Perceived crisis and reforms: issues, explanations, and remedies. *Psychological Bulletin, 144*(7), 757-777. doi:10.1037/bul0000154

de Diana, I., & van Schaik, P. (1993). Courseware engineering outlined: an overview of some research issues. *Educational and Training Technology International, 30*(3), 191-211. doi:10.1080/0954730930300302

de Jong, T., Weinberger, A., Girault, I., Kluge, A., Lazonder, A. W., Pedaste, M., . . . Zacharia, Z. C. (2012). Using scenarios to design complex technology-enhanced learning environments. *Educational Technology Research and Development, 60*(5), 883-901. doi:10.1007/s11423-012-9258-1

Dennehy, D., & Conboy, K. (2017). Going with the flow: An activity theory analysis of flow techniques in software development. *Journal of Systems and Software, 133*, 160-173. doi:10.1016/j.jss.2016.10.003

Depradine, C., & Gay, G. (2004). Active participation of integrated development environments in the teaching of object-oriented programming. *Computers & Education, 43*(3), 291-298. doi:10.1016/j.compedu.2003.10.009

Dixon-Woods, M., Cavers, D., Agarwal, S., Annandale, E., Arthur, A., Harvey, J., . . . Sutton, A. J. (2006). Conducting a critical interpretive synthesis of the literature on access to healthcare by vulnerable groups. *BMC Medical Research Methodology, 6*(1), 35. doi:10.1186/1471-2288-6-35

Domínguez, A., Saenz-de-Navarrete, J., de-Marcos, L., Fernández-Sanz, L., Pagés, C., & Martínez-Herráiz, J.-J. (2013). Gamifying learning experiences: Practical implications and outcomes. *Computers & Education, 63*, 380-392. doi:10.1016/j.compedu.2012.12.020

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput., 5*(3), 4. doi:10.1145/1163405.1163409

Educational Technology Research and Development. *Educational technology research and development*. Retrieved from https://www.springer.com/education+%26+language/learning+%26+instruction/journal/11423

Elo, S., Kääriäinen, M., Kanste, O., Pölkki, T., Utriainen, K., & Kyngäs, H. (2014). Qualitative content analysis: a focus on trustworthiness. *SAGE Open, 4*(1), 2158244014522633. doi:10.1177/2158244014522633

Engeström, Y. (1987). *Learning by expanding: An activity-theoretical approach to developmental research* (2nd ed.). Cambridge, MA: Cambridge University Press.

Engeström, Y. (1996). Developmental work research as educational research: looking ten years back and into the zone of proximal development. *Nordisk pedagogik, 16*(3), 131-143.

Engeström, Y. (2000). Activity theory as a framework for analyzing and redesigning work. *Ergonomics, 43*(7), 960-974. doi:10.1080/001401300409143

Engestrom, Y., Miettinen, R., & Punamaki, R.-l. (1999). *Perspectives on activity theory* (Y. Engeström, R. Miettinen, & R.-L. Punamäki Eds.). Cambridge, MA: Cambridge University Press.

Finn, J. D. (1953). Professionalizing the audio-visual field. *Audio Visual Communication Review, 1*(1), 6-17.

Fishman, B., Marx, R., Blumenfeld, P., Krajcik, J., & Soloway, E. (2004). Creating a framework for research on systemic technology innovations. *The Journal of the Learning Sciences, 13*(1), 43-76. doi:10.1207/s15327809jls1301_3

Flemming, K. (2010). Synthesis of quantitative and qualitative research: an example using Critical Interpretive Synthesis. *Journal of Advanced Nursing, 66*(1), 201-217. doi:10.1111/j.1365-2648.2009.05173.x

Forward, A., & Lethbridge, T. C. (2002). *The relevance of software documentation, tools and technologies: A survey*. Paper presented at the Proceedings of the 2002 ACM symposium on Document engineering, McLean, Virginia, USA.

Fowler, M. (2018). *Refactoring: improving the design of existing code*: Addison-Wesley Professional.

Gage, N. A., & Stevens, R. N. (2018). Rigor, replication, and reproducibility: Increasing the relevance of behavioral disorders research. *Education & Treatment of Children, 41*(4), 567-588. doi:10.1353/etc.2018.0029

Gagné, R. M., Briggs, L. J., & Wager, W. W. (1991). *Principles of instructional design* (4th ed.). Forth Worth, TX: Harcourt Brace Jovanovich College Pub.

Gaudel, M. C. (1981). Compiler generation from formal definition of programming languages: a survey. In *Lecture Notes in Computer Science* (Vol. 107, pp. 96-114). Berlin, Heidelberg: Springer Berlin Heidelberg.

Gaydos, M. (2015). Seriously considering design in educational games. *Educational Researcher, 44*(9), 478-483. doi:10.3102/0013189x15621307

Getenet, S. (2019). Using design-based research to bring partnership between researchers and practitioners. *Educational Research, 61*(4), 482.

Hadjistassou, S. K. (2012). An activity theory exegesis on conflict and contradictions in networked discussions and feedback exchanges. *CALICO Journal, 29*(2), 367-388.

Hirumi, A., Kleinsmith, A., Johnsen, K., Kubovec, S., Eakins, M., Bogert, K., . . . Cendan, J. (2016). Advancing virtual patient simulations through design research and interPLAY: part I: design and development. *Educational Technology Research and Development, 64*(4), 763-785. doi:10.1007/s11423-016-9429-6

Huang, Y.-M., & Huang, Y.-M. (2015). A scaffolding strategy to develop handheld sensor-based vocabulary games for improving students' learning motivation and performance. *Educational Technology Research and Development, 63*(5), 691-708. doi:10.1007/s11423-015-9382-9

Hucka, M., & Graham, M. J. (2018). Software search is not a science, even among scientists: A survey of how scientists and engineers find software. *Journal of Systems and Software, 141*, 171-191. doi:10.1016/j.jss.2018.03.047

International Journal of Designs for Learning. *Submissions*. Retrieved from https://scholarworks.iu.edu/journals/index.php/ijdl/about/submissions#authorGuidelines

Iyamu, T., & Shaanika, I. (2019). The use of activity theory to guide information systems research. *Education & Information Technologies, 24*(1), 165-180. doi:10.1007/s10639-018-9764-9

Jackson, S., & Brannon, S. (2018). In-house software development: Considerations for implementation. *The Journal of Academic Librarianship*, *44*(6), 689-691. doi:10.1016/j.acalib.2018.10.008

Kali, Y., & Ronen-Fuhrmann, T. (2011). Teaching to design educational technologies. *International Journal of Learning Technology, 6*(1), 4-23. doi:10.1504/IJLT.2011.040147

Kangas, M., Koskinen, A., & Krokfors, L. (2017). A qualitative literature review of educational games in the classroom: the teacher’s pedagogical activities. *Teachers and Teaching, 23*(4), 451-470. doi:10.1080/13540602.2016.1206523

Kankuzi, B., & Sajaniemi, J. (2016). A mental model perspective for tool development and paradigm shift in spreadsheets. *International Journal of Human-Computer Studies, 86*, 149-163. doi:10.1016/j.ijhcs.2015.10.005

Karanasios, S., Thakker, D., Lau, L. L., Allen, D., Dimitrova, V., & Norman, A. (2013). Making sense of digital traces: an activity theory driven ontological approach. *Journal of the American Society for Information Science & Technology, 64*(12), 2452-2467. doi:10.1002/asi.22935

Karasavvidis, I. (2009). Activity Theory as a conceptual framework for understanding teacher approaches to Information and Communication Technologies. *Computers & Education, 53*(2), 436-444. doi:10.1016/j.compedu.2009.03.003

Karasavvidis, I. (2010). Understanding wikibook-based tensions in higher education: An activity theory approach. *E-Learning and Digital Media, 7*(4), 386-394. doi:10.2304/elea.2010.7.4.386

Kernebeck, U. (1997). Component libraries for software re-use. *Microprocessors and Microsystems, 21*(1), 49-54. doi:10.1016/S0141-9331(97)00019-7

Khan, S. (2008). The case in case-based design of educational software: A methodological interrogation. *Educational Technology Research and Development, 56*(4), 423-447. doi:10.1007/s11423-006-9028-z

Kirschner, P. A. (2004). Design, development, and implementation of electronic learning environments for collaborative learning. *Educational Technology Research and Development, 52*(3), 39-46. doi:10.1007/BF02504674

Kneuper, R. (2017). Sixty years of software development life cycle models. *IEEE Annals of the History of Computing, 39*(3), 41-54. doi:10.1109/MAHC.2017.3481346

Ko, A. J., DeLine, R., & Venolia, G. (2007). *Information needs in collocated software development teams.* Paper presented at the International conference on software engineering, Washington, DC.

Korpela, M., Mursu, A., & Soriyan, H. A. (2002). Information systems development as an activity. *Computer Supported Cooperative Work, 11*(1), 111-128. doi:10.1023/A:1015252806306

Kultur, C., Oytun, E., Cagiltay, K., Ozden, M. Y., & Kucuk, M. E. (2004). *Moving toward scorm compliant content production at educational software company: Technical and administrative challenges.* Paper presented at the Association for Educational Communications and Technology Annual Meeting.

Laine, T. H., Nygren, E., Dirin, A., & Suk, H.-J. (2016). Science spots ar: a platform for science learning games with augmented reality. *Educational Technology Research and Development, 64*(3), 507-531. doi:10.1007/s11423-015-9419-0

Lamb, R., & Johnson, S. (2006). Social aspects of digital information in perspective: introduction to a special issue. *Journal of Digital Information, 5*(4).

Leenaars, F. A. J., van Joolingen, W. R., Gijlers, H., & Bollen, L. (2014). Gearsketch: an adaptive drawing-based learning environment for the gears domain. *Educational Technology Research and Development, 62*(5), 555-570. doi:10.1007/s11423-014-9345-6

Leont'ev, A. N. (1974). The problem of activity in psychology. *Soviet psychology, 13*(2), 4-33. doi:10.2753/RPO1061-040513024

Leupers, R. (2002). Compiler design issues for embedded processors. *IEEE Design & Test of Computers, 19*(4), 51-58. doi:10.1109/MDT.2002.1018133

Lewin, C., Cranmer, S., & McNicol, S. (2018). Developing digital pedagogy through learning design: an activity theory perspective. *British Journal of Educational Technology, 49*(6), 1131-1144. doi:10.1111/bjet.12705

Liberati, A., Altman, D. G., Tetzlaff, J., Mulrow, C., Gøtzsche, P. C., Ioannidis, J. P. A., . . . Moher, D. (2009). The prisma statement for reporting systematic reviews and meta-analyses of studies that evaluate health care interventions: explanation and elaboration. *Annals of Internal Medicine, 151*(4), W-65-W-94. doi:10.1371/journal.pmed.1000100

Liu, J. (2013). The assessment agent system: design, development, and evaluation. *Educational Technology Research and Development, 61*(2), 197-215. doi:10.1007/s11423-013-9286-5

Loui, M. C. (2015). Replication, taxonomy, and thanks. *Journal of Engineering Education, 104*(4), 363-363. doi:10.1002/jee.20103

Maher, J. H., & Ingram, A. L. (1989). *Software engineering and isd: Similarities, complementarities, and lessons to share*. Paper presented at the Association for Educational Communications and Technology, Dallas, TX.

Mahmoud, A., & Williams, G. (2016). Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering, 21*(3), 357. doi:10.1007/s00766-016-0252-8

Marcus, A., & Maletic, J. I. (2003). *Recovering documentation-to-source-code traceability links using latent semantic indexing*. Paper presented at the Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon.

Margulieux, L., Ketenci, T. A., & Decker, A. (2019). Review of measurements used in computing education research and suggestions for increasing standardization. *Computer Science Education, 29*(1), 49-78. doi:10.1080/08993408.2018.1562145

Matthews, T., Rattenbury, T., & Carter, S. (2007). Defining, designing, and evaluating peripheral displays: an analysis using activity theory. *Human-Computer Interaction, 22*(1-2), 221-261.

McBurney, P. W., & McMillan, C. (2014). *Automatic documentation generation via source code summarization of method context*. Paper presented at the Proceedings of the 22nd International Conference on Program Comprehension, Hyderabad, India.

McKenney, S., & Reeves, T. C. (2013). Systematic review of design-based research progress: Is a little knowledge a dangerous thing? *Educational Researcher, 42*(2), 97-100. doi:10.3102/0013189x12463781

Mertz, M., Kahrass, H., & Strech, D. (2016). Current state of ethics literature synthesis: a systematic review of reviews. *BMC Medicine, 14*, 1-12. doi:10.1186/s12916-016-0688-1

Modell, M. G. (2014). Iterating over a method and tool to facilitate equitable assessment of group work. *International Journal of Designs for Learning, 4*(1), 39-53.

Moreno, L. (2014). *Summarization of complex software artifacts*. Paper presented at the Companion Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India.

Morrison, L., Yardley, L., Powell, J., & Michie, S. (2012). What design features are used in effective e-health interventions? A review using techniques from critical interpretive synthesis. *Telemedicine and e-Health, 18*(2), 137-144. doi:10.1089/tmj.2011.0062

Myers, G. J., Badgett, T., Thomas, T. M., & Sandler, C. (2004). *The art of software testing* (Vol. 2): Wiley Online Library.

Neale, J. (2016). Iterative categorization (ic): a systematic technique for analysing qualitative data. *Addiction, 111*(6), 1096-1106. doi:10.1111/add.13314

Nielsen, C., & Cappelen, K. (2014). Exploring the mechanisms of knowledge transfer in university-industry collaborations: A study of companies, students and researchers. *Higher Education Quarterly, 68*(4), 375-393. doi:10.1111/hequ.12035

Oliver, M. (2011). Technological determinism in educational technology research: some alternative ways of thinking about the relationship between learning and technology. *Journal of Computer Assisted Learning, 27*(5), 373-384. doi:10.1111/j.1365-2729.2011.00406.x

Oncu, S., & Cakir, H. (2011). Research in online learning environments: Priorities and methodologies. *Computers & Education, 57*(1), 1098-1108. doi:10.1016/j.compedu.2010.12.009

Ormel, B. J. B., Pareja Roblin, N. N., McKenney, S. E., Voogt, J. M., & Pieters, J. M. (2012). Research–practice interactions as reported in recent design studies: still promising, still hazy. *Educational Technology Research and Development, 60*(6), 967-986. doi:10.1007/s11423-012-9261-6

Paek, S., Hoffman, D. L., & Black, J. B. (2016). Perceptual factors and learning in digital environments. *Educational Technology Research and Development, 64*(3), 435-457. doi:10.1007/s11423-016-9427-8

Panke, S., Kohls, C., & Gaiser, B. (2007). Participatory development strategies for open source content management systems. *Innovate: Journal of Online Education, 3*(2), 8.

Park, S. W., & Kim, C. M. (2014). Virtual Tutee System: a potential tool for enhancing academic reading engagement. *Educational Technology Research and Development, 62*(1), 71-97. doi:10.1007/s11423-013-9326-1

Park, S. W., & Kim, C. M. (2016). The effects of a virtual tutee system on academic reading engagement in a college classroom. *Educational Technology Research and Development, 64*(2), 195-218. doi:10.1007/s11423-015-9416-3

Park, T., & Lim, C. (2019). Design principles for improving emotional affordances in an online learning environment. *Asia Pacific Education Review, 20*(1), 53-67. doi:10.1007/s12564-018-9560-7

Parmaxi, A., & Zaphiris, P. (2015). Developing a framework for social technologies in learning via design-based research. *Educational Media International, 52*. doi:10.1080/09523987.2015.1005424

Peña-Ayala, A., Sossa, H., & Méndez, I. (2014). Activity theory as a framework for building adaptive e-learning systems: a case to provide empirical evidence. *Computers in Human Behavior, 30*, 131-145. doi:10.1016/j.chb.2013.07.057

Penuel, W. R., Fishman, B. J., Cheng, B. H., & Sabelli, N. (2011). Organizing research and development at the intersection of learning, implementation, and design. *Educational Researcher, 40*(7), 331-337. doi:10.3102/0013189x11421826

Persky, A. M., & Murphy, K. (2019). Investigating whether transfer of learning in pharmacy students depends more on knowledge storage or accessibility. *American Journal of Pharmaceutical Education, 83*(6), 1274-1281.

Plonsky, L. (2015). Quantitative considerations for improving replicability in call and applied linguistics. *CALICO Journal, 32*(2), 232-244. doi:10.1558/cj.v32i2.26857

Plummer, M. (2018). Lived experiences of grooming among australian male survivors of child sexual abuse. *Journal of Interpersonal Violence, 33*(1), 37.

Pool, J., & Laubscher, D. (2016). Design-based research: is this a suitable methodology for short-term projects? *Educational Media International, 53*(1), 42-52. doi:10.1080/09523987.2016.1189246

Porte, G. (2013). Who needs replication? *CALICO Journal, 30*(1), 10-15. doi:10.11139/cj.30.1.10-15

Postholm, M. B. (2015). Methodologies in cultural–historical activity theory: The example of school-based development. *Educational Research, 57*(1), 43-58. doi:10.1080/00131881.2014.983723

Randolph, J., J. (2009). A guide to writing the dissertation literature review. *Practical Assessment, Research & Evaluation, 14*(13). doi:https://doi.org/10.7275/b0az-8t74

Researcher, E. *Educational researcher*. Retrieved from https://us.sagepub.com/en-us/nam/journal/educational-researcher

Richey, R. C. (1997). Research on instructional development. *Educational Technology Research and Development*, *45*(3), 91-100. doi:https://doi.org/10.1007/BF02299732

Robillard, P. N. (1989). Automating comments. *ACM SIGPLAN Notices, 24*(5), 66-70. doi:10.1145/66068.66073

Roschelle, J., & DiGiano, C. (2004). Escot: Coordinating the influence of r&d and classroom practice to produce educational software from reusable components. *Interactive Learning Environments, 12*(1-2), 73-107. doi:10.1080/1049482042000300904

Salas-Morera, L., Arauzo-Azofra, A., García-Hernández, L., Palomo-Romero, J. M., & Hervás-Martínez, C. (2013). PpcProject: An educational tool for software project management. *Computers & Education, 69*, 181-188. doi:10.1016/j.compedu.2013.07.018

Sanders, N. E., Faesi, C., & Goodman, A. A. (2014). A new approach to developing interactive software modules through graduate education. *Journal of Science Education and Technology, 23*(3), 431-440. doi:10.1007/s10956-013-9474-4

Sannino, A., & Engeström, Y. (2018). Cultural-historical activity theory: founding insights and new challenges. *Cultural-Historical Psychology, 14*(3), 43-56. doi:10.17759/chp.2018140304

Schuch, D. (2001). The research assistant. *TechTrends*, *45*(2), 17-18. doi:10.1007/bf02763493

Sembrat, E. S. (2020). A review of the literature on process at the nexus of instructional and software design *Manuscript in preparation.*

Shavelson, R., Phillips, D. C., Towne, L., & Feuer, M. (2005). On the science of education design studies. *Educational Researcher, 32*(1), 25-28. doi:10.3102/0013189X032001025

Smith, P. L., & Ragan, T. J. (2005). *Instructional design*: John Wiley & Sons.

Sonya Zhang, X., & Dorn, B. (2012). Accelerating software development through agile practices-a case study of a small-scale, time-intensive web development project at a college-level it competition. *Journal of Information Technology Education, 11*. doi:10.28945/1545

Sun, H.-M., & Cheng, W.-L. (2009). The input-interface of Webcam applied in 3D virtual reality systems. *Computers & Education, 53*(4), 1231-1240. doi:10.1016/j.compedu.2009.06.006

Sun, P. Y.-T., & Scott, J. L. (2005). An investigation of barriers to knowledge transfer. *Journal of Knowledge Management, 9*(2), 75-90. doi:10.1108/13673270510590236

Tan, L., Yuan, D., Krishna, G., & Zhou, Y. (2007). /*icomment: bugs or bad comments?*. *SIGOPS Oper. Syst. Rev., 41*(6), 145-158. doi:10.1145/1323293.1294276

Tergan, S.-O. (2006). Checklists for the evaluation of educational software: Critical review and prospects. *Innovations in Education and Training International, 35*(1), 9-20. doi:10.1080/1355800980350103

The Design-Based Research Collective. (2003). Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher, 32*(1), 5-8. doi:10.3102/0013189x032001005

Toader, A. F., & Kessler, T. (2018). Task variation and mental models divergence influencing the transfer of team learning. *Small Group Research, 49*(5), 545-575. doi:10.1177/1046496418786429

Tripp, S. D., & Bichelmeyer, B. (1990). Rapid prototyping: An alternative instructional design strategy. *Educational Technology Research and Development, 38*(1), 31-44. doi:10.1007/bf02298246

Trust, T. (2017). Using cultural historical activity theory to examine how teachers seek and share knowledge in a peer-to-peer professional development network. *Australasian Journal of Educational Technology, 33*(1), 98-113. doi:10.14742/ajet.2593

Uden, L., Valderas, P., & Pastor, O. (2008). An activity-theory-based model to analyse web application requirements. *Information Research, 13*(2). doi:10.1016/j.compedu.2015.03.023

van der Schaaf, M., Donkers, J., Slof, B., Moonen-van Loon, J., van Tartwijk, J., Driessen, E., . . . Ten Cate, O. (2017). Improving workplace-based assessment and feedback by an e-portfolio enhanced with learning analytics. *Educational Technology Research and Development, 65*(2), 359-380. doi:10.1007/s11423-016-9496-8

Vanderhoven, E., Schellens, T., Vanderlinde, R., & Valcke, M. (2016). Developing educational materials about risks on social network sites: a design based research approach. *Educational Technology Research and Development, 64*(3), 459-480. doi:10.1007/s11423-015-9415-4

Veletsianos, G., Beth, B., Lin, C., & Russell, G. (2016). Design principles for thriving in our digital world: A high school computer science course. *Journal of Educational Computing Research, 54*(4), 443-461. doi:10.1177/0735633115625247

Verdú, E., Regueras, L. M., Gal, E., de Castro, J. P., Verdú, M. J., & Kohen-Vacs, D. (2017). Integration of an intelligent tutoring system in a course of computer network design. *Educational Technology Research and Development, 65*(3), 653-677. doi:10.1007/s11423-016-9503-0

von Wangenheim, C. G., Hauck, J. C. R., Demetrio, M. F., Pelle, R., Cruz Alves, N., Barbosa, H., & Azevedo, L. F. (2018). Codemaster - automatic assessment and grading of app inventor and snap! programs. *Informatics in Education, 17*(1), 117-150. doi:10.15388/infedu.2018.08

Vygotsky, L. (1978). *Mind in society: The development of higher psychological processes.* Cambridge, MA: Harvard University Press.

Wade-Jaimes, K., Cohen, J. D., & Calandra, B. (2019). Mapping the evolution of an after-school STEM club for African American girls using activity theory. *Cultural Studies of Science Education, 14*(4), 981-1010. doi:10.1007/s11422-018-9886-9

Waight, N., Liu, X., & Gregorius, R. M. (2015). Understanding the life cycle of computer-based models: the role of expert contributions in design, development and implementation. *Educational Technology Research and Development, 63*(6), 831-859. doi:10.1007/s11423-015-9402-9

Wang, Q., Nieveen, N., & van den Akker, J. (2007). Designing a computer support system for multimedia curriculum development in Shanghai. *Educational Technology Research and Development, 55*(3), 275-295. doi:10.1007/s11423-006-9017-2

Wei, X., Weng, D., Liu, Y., & Wang, Y. (2015). Teaching based on augmented reality for a technical creative design course. *Computers & Education, 81*, 221-234. doi:10.1016/j.compedu.2014.10.017

Wells, G. (2002). The role of dialogue in activity theory. *Mind, Culture, and Activity, 9*(1), 43-66. doi:10.1207/S15327884MCA0901_04

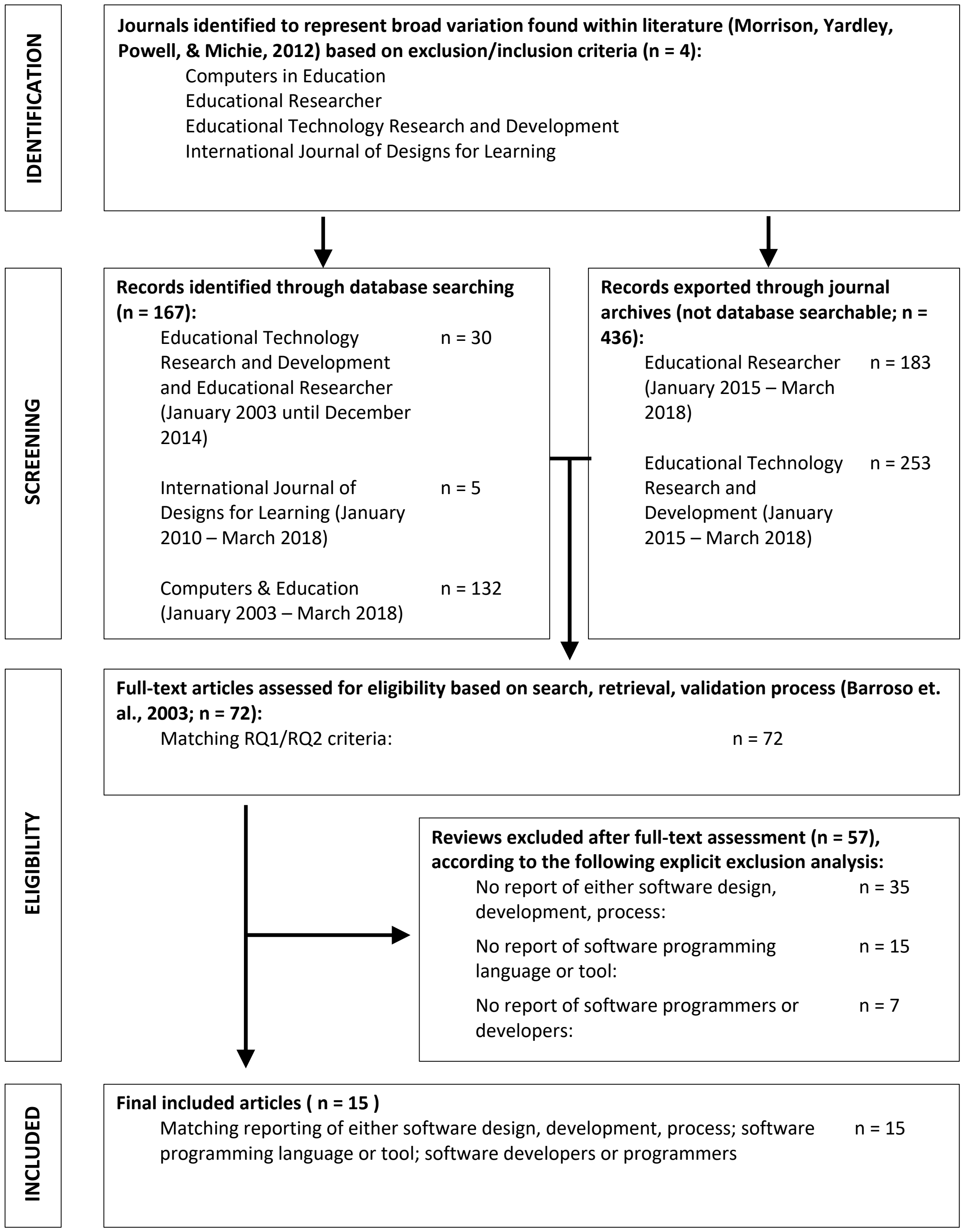
Winters, N., & Mor, Y. (2008). IDR: A participatory methodology for interdisciplinary design in technology enhanced learning. *Computers & Education, 50*(2), 579-600. doi:10.1016/j.compedu.2007.09.015

Wolgemuth, J. R., Hicks, T., & Agosto, V. (2017). Unpacking assumptions in research synthesis: A critical construct synthesis approach. *Educational Researcher, 46*(3), 131-139. doi:10.3102/0013189x17703946

Zawacki-Richter, O., & Naidu, S. (2016). Mapping research trends from 35 years of publications in Distance Education. *Distance Education, 37*(3), 245-269. doi:10.1080/01587919.2016.1185079

Zhu, H. (2005). *Software design methodology: From principles to architectural styles*. Oxford: Butterworth-Heinemann.

Appendix


IDENTIFICATION
Journals identified to represent broad variation found within literature (Morrison, Yardley, Powell, & Michie, 2012) based on exclusion/inclusion criteria (n = 4):
Computers in Education
Educational Researcher
Educational Technology Research and Development
International Journal of Designs for Learning
SCREENING
Records identified through database searching (n = 167):
Educational Technology Research and Development and Educational Researcher (January 2003 until December 2014) n = 30
International Journal of Designs for Learning (January 2010 – March 2018) n = 5
Computers & Education (January 2003 – March 2018) n = 132
Records exported through journal archives (not database searchable; n = 436):
Educational Researcher (January 2015 – March 2018) n = 183
Educational Technology Research and Development (January 2015 – March 2018) n = 253
ELIGIBILITY
Full-text articles assessed for eligibility based on search, retrieval, validation process (Barroso et. al., 2003; n = 72):
Matching RQ1/RQ2 criteria: n = 72
Reviews excluded after full-text assessment (n = 57), according to the following explicit exclusion analysis:
No report of either software design, development, process: n = 35
No report of software programming language or tool: n = 15
No report of software programmers or developers: n = 7
INCLUDED
Final included articles (n = 15)
Matching reporting of either software design, development, process; software programming language or tool; software developers or programmers n = 15