Int J Artif Intell Educ (2016) 26:37–71
DOI 10.1007/s40593-015-0076-6

ARTICLE

Coordinating the Complexity of Tools, Tasks, and Users: On Theory-based Approaches to Authoring Tool Usability

Tom Murray[1]

Published online: 5 November 2015

Abstract Intelligent Tutoring Systems authoring tools are highly complex educational software applications used to produce highly complex software applications (i.e. ITSs). How should our assumptions about the target users (authors) impact the design of authoring tools? In this article I first reflect on the factors leading to my original 1999 article on the state of the art in ITS authoring tools and consider some challenges facing authoring tool researchers today. Then, in the bulk of the paper, I propose some principled foundations for future authoring tool design, focusing on operationalizing the construct of complexity—for tool, task, and user. ITS authoring tools are major undertakings and to redeem this investment it is important to anticipate actual user needs and capacities. I propose that one way to do this is to match the complexity of tool design to the complexity of authoring tasks and the complexity capacity of users and user communities. Doing so entails estimating the complexity of the mental models that a user is expected to build in order to use a tool as intended. The goal is not so much to support the design of more powerful authoring tools as it is to design tools that meet the needs of realistic user audiences. This paper presents some exploratory ideas on how to operationalize the concept of complexity for tool, task, and user. The paper draws from the following theories and frameworks to weave this narrative: Complexity Science, Activity Theory, Epistemic Forms and Games, and adult cognitive developmental theory (Hierarchical Complexity Theory). This exploration of usability and complexity is applicable to the design of any type of complex authoring application, though the application area that motivated the exploration is ITS authoring.

A shorter version of this article appears as a chapter in "Theory-based Authoring Tool Design: Considering the Complexity of Tasks and Mental Models", Chapter 2 in R. Sottilare, A. Graesser, X. Hu, & K. Brawner (Eds), Design Recommendations For Intelligent Tutoring Systems: Volume 3 - Authoring Tools and Expert Modeling Techniques (2015).

✉ Tom Murray
tommurray.us@gmail.com

[1] School of Computer Science, University of Massachusetts, Amherst, MA, USA

Keywords Authoring tools · Intelligent tutoring systems · Complexity science · Epistemic forms · Activity theory · Hierarchical complexity theory

Introduction

In 1999 I wrote an article on the "state of the art" in ITS authoring tools R&D, and in 2003 I co-edited a book on that subject with Sharon Ainsworth and Stephen Blessing. Apparently the state of the art paper (Murray 1999, and its update in Murray 2003a) has been cited many times, prompting the managing editors of this journal to ask me, along with authors of other "classic" IJAIED papers, to write a reflective piece about that paper and its topic for a special issue of the journal. I baulked at first because, though I have continued to build in-house authoring tools for all of my projects over succeeding years (it adds up to about 10 of them), it has been some time since I considered myself doing research on that subject and have not been keeping up with the associated literature. I was assured by the editors that the purpose was not to create a revised updated analysis of the state of the art, but rather was invited to be somewhat of a storyteller about my journey to and after the original paper and to reflect in any way that seemed fit on the topic in the contemporary context (for more recent work in the field, see Aleven and Sewall 2010; Cristea 2005; Olsen et al. 2013; Specht 2012; Suraweera et al. 2010; Mitrovic et al. 2009; Sottilare et al. 2012, 2014; Razzaq et al. 2009; Aleven et al. 2015; Ritter 2015).

As I thought about it realized that I did indeed have some thoughts on the subject that reflect what I have learned on my winding interdisciplinary scholarly path over the last decade. The authoring tools I have been building of late are for myself and research collaborators for specific purposes (to visualize, edit, verify, and analyze complex information in data-driven instructional systems), but for "real" authoring tool projects the question that seems to perpetually haunt is "who are going to use these tools and how do we ensure that the tools meet end user needs?" My approach to answering this question spirals around the theme of *complexity*. It turns out complexity is a complex topic, especially for one with an interdisciplinary lens, and my musings are rather longer than the Editors in Chief might have expected in a retrospective article. They will think twice before asking me again in another 10 yeas. But before going any further I will start with a short introduction to the field of ITS authoring systems.

ITS Authoring Tool Design Tradeoffs Intelligent Tutoring Systems (ITSs) are highly complex educational software applications (or learning environments) that can include these components: User Interface (which might include a simulated phenomena or task environment), Expert Knowledge Model (of the task and/or knowledge), Learner Knowledge Model, Pedagogical Model, and Curriculum Model (also collaborative learning environments may include group-level aspects of any of these) (see Woolf 2010). For several decades developers and researchers have been investigating the possibilities for creating ITS authoring tools because these are hoped to (1) reduce the effort and cost of building or customizing ITSs, and (2) allow non-programmers, including teachers and domain experts (and even students) to participate fully or partly in building or customizing ITSs (Murray et al. 2003; Aleven et al. 2006; Suraweera et al. 2010; Constantin et al. 2013; Ainsworth et al. 2003; Ritter and Blessing 1998).

There are many design tradeoffs involved—the primary one being that in general the easier or more efficient a tool is to use, the more simplistic or constrained are the ITSs that can be built from it.[1] Trivial examples at two extremes are: a tool that allows the author to select among checkboxes and lists to order and toggle and sequence features and curriculum items in an otherwise fixed system; vs. a tool that is so complicated and multi-featured that building an ITS with it is not much easier than traditional software programming. One can imagine a design tradeoff space among *usability*, *depth*, *and flexibility* (see Murray 2004). Depth, which refers to the structural or casual depth of any of the ITS sub-models, is usually at odds with flexibility, which is the ability to author a diversity of types of ITSs. Usability is usually at odds with both depth and flexibility, i.e. a system that facilitates building deep models or many types of models tends to be more powerful yet less usable. A main theme of this article will be to provide some rough metrics to help with these design tradeoffs.

Towards Theoretical Foundations Unlike educational software, whose user audience is relatively well defined and known, target users of authoring tools are less well defined and understood (unless the tool is intended for in-house use by a few specialized personnel). We can draw from the standard literature on usability and Human-Computer Interaction (including user-participatory design), for tool design principles, but in addition there are some issues specific to authoring tools (of any sort, not just for ITSs) that I find quite interesting. Influenced by topics I have studied since my early papers on the subject, I have come to believe that a key issue is in *how one matches the complexity of the authoring task to the complexity of a tool and the complexity-capacity of the target user*. Thus, in the bulk of this paper I will sketch some preliminary considerations and principles that are intended to initiate inquiry in this direction. The suggestions are speculative, and are motivated by my belief that the set of theoretical frameworks I will weave together are useful, unexplored within the ITS community, and have not been integrated in any prior work.

Taking a theoretical approach to ITS (or any) authoring tool usability is rarely done, and risks being too theoretical for a field with such practical goals, but my aim here is to point toward possible theoretical foundations for the (sub-) field. "Theory" can sometimes refer to a mere conceptual framework (without any underlying causal theory), but here I mean cognitive, social, epistemological, and/or information science theories that provide theoretical underpinnings. Foundational theories (especially the Learning and Cognitive Sciences) are now routinely considered in the design of ITSs and other educational software, but are rarely brought into discussions about the design or use of authoring tools. Note however that the goal is not to produce a unified "theory" of authoring tool usability, but to take a novel look at how one can base design usability principles upon several theoretical frameworks that seem relevant but have not been linked with ITS authoring as yet.[2]

[1] These are generic tradeoffs, all other things begin equal. But note that complex systems can be constructed using relatively simple tools, as in the CTAT system (Aleven et al., in submission). Representational formalisms that well match the domain and task needs will go a long way towards making authoring easier without compromising the complexity or scope of the resulting ITSs.

[2] Learning theories can and should be brought to bear on ITS authoring tool design, for example to constrain the design of ITSs to adhere to known principles of learning (thanks to comments from Vincent Aleven and Steve Ritter for pointing this out). My focus here is on the more general theme of tool complexity and usability, which is not concerned with the content or model used by the tutoring system.

Design Science and Usability Theory draw on socio-cognitive theories to explore the relationships between the design of artifacts and the needs, capabilities, and limitations of intended users (and other stakeholders) (see Oja 2010; Norman 1988; Nielsen 1993). Originally these theories were in response to the (now more accepted) realization that domain experts (those who are not instructors), traditional software architects, and academics all historically have difficulty predicting or imagining the needs and limitations of the average software user and the average real-life task scenario (or difficulty predicting the *range* of users and task scenarios). Thus software design, and artifact design in general, is increasingly understood as needing (1) empirical trial-and-error development, (2) the skills of rigorous empathy and imagination to put oneself in the shoes of a range of types of users and situations, and (3) some basis in underlying psycho-socio-technical theory (Brown and Campione 1996; Cobb et al. 2003).

The notion of assessing and coordinating complexity among tool, task, and user is a central theme in this particular theoretical exploration. In what follows, I will first reflect on the factors leading to my 1999 article on authoring tools. I will then consider some challenges facing authoring tool researchers today. Then, in the bulk of the paper, I will propose some theoretical foundations for future authoring tool design. I will draw from the following theories and frameworks to weave this particular theoretical narrative:

- Complexity in software design
- Activity Theory
- Epistemic Forms and Games, and
- Adult cognitive developmental theory (i.e. Hierarchical Complexity Theory).

Theories of complex software design will be used to emphasize some of the issues, because ITS authoring tools are *complex artifacts designed to produce complex artifacts*. Complexity Science will also help us operationalize what is meant by complexity in general. Activity Theory, which highlights the relationships between an artifact and its usage-*tasks*, usage-*rules*, and *community* of practice, will provide an orientation and basic vocabulary for the task of ITS design by various types of users in an authoring role. We can ask whether a tool and its "rules" of use afford the accomplishment of a particular task for a particular class of users. Much of the process of matching tool/task complexity to user (and community) complexity capacity revolves around the complexity of the *mental models* that a user is expected to build in order to use a tool as intended. Collins's work on *Epistemic Forms and Games* provides a highly useful framework for talking about this tool-rule-user match in holistic terms at the right level of granularity (Collins and Ferguson 1993). At this point we will have a framework for describing many *sources* of complexity in tools, tasks, and users (cognition or mental models), but no good way to order or coordinate these types of complexity. For that we will draw on Hierarchical Complexity Theory and related theories of adult cognitive development to suggest this order as a final step in matching the complexity of an authoring tool to the complexity capacity of its target users (Commons and Richards 1984; Fischer 1980).

Overview Following is an extended overview of the narrative arc of the paper.
Framing the context:

- As a **prelude** I describe my history with the subject of authoring tools, and then outline some challenges facing authoring tool designers and researchers today.
- How can we decide whether it is advantageous to build an authoring tool (vs. building ITSs from scratch) and what features to include in an authoring tool? I frame questions about ITS authoring tool design in terms of **design decisions and tradeoffs**.
- In designing ITS authoring tools there are a number of design tradeoffs that can be summarized in terms of finding the right balance between **usability** (simplicity and efficiency of use) and **power** (flexibility/breadth and depth) for the intended author community and the end-product educational software systems.
- Authoring tools are quite labor-intensive to build, and yet the realistic demand (in terms of markets or needs for building many ITSs) and the realistic availability of ITS authors and designers (e.g. the time-availability of teachers and experts to learn and use complex tools) is limited. Authoring tools may be practical only for domains with high demand and reasonably established pedagogy (such as mathematics).[3] (However, increasingly learning and instruction are happening online, which argues for an increasing demand in general.) We can speak of finding the sweet spot that balances the **cost** of investing in authoring tool construction vs. the "**risk**" that the investment will not be worth it. This is also about matching the *vision* of a powerful and useful tool with the likely *reality* awaiting its release.
- Classical design and usability theory applies to all of the tradeoffs mentioned above. Specifically, the principle to "match between system and the real world" speaks to using vocabulary, mental models, and task-demands that users already have (or can easily learn). ITS authoring tools are complex systems created to build complex systems. Though there are many design decisions to make, I propose that the overall **lens of "complexity**" is most useful, and informs the principles given in classical usability theory. In a very general sense, complexity tends to increase with power and decrease with usability—and we are concerned with balancing complexity (which supports more power) with usability. That is, for tradeoffs related to usability, power, cost, risk, mental models, etc., we can speak in general of matching the level of complexity of the tool with the "complexity capacity" of those expected to use the tool.
- The bulk of the paper explores a diverse collection of frameworks that are woven together to answer the question of **how one might go about specifying the complexity** of the system and the complexity capacity of intended users. This ensemble of frameworks is only a beginning step pointing to additional work that would need to be done to operationalize their implementation.
- **Complexity Science** (including Information Theory) can be used to characterize the complexity of software artifacts (i.e. authoring tools) in terms of the quantity

[3] There is of course much controversy about how to teach mathematics, and no consensus on pedagogy in any area. However, it seems more likely for mathematics (and perhaps physical sciences and basic computer programming), vs. many other domains, that content map and a pedagogical approach can be created that is acceptable to many.

and variety of parts and interactions within a system. Similar methods can be used to characterize the complexity of a task in terms of the quantity and variety of steps and decisions involved. "Dynamic complexity" must also be taken into account in authoring tools, as authors are tasked with building *and debugging* systems that run or "behave" according to a specification. ITS authors are not only "**constructing**" artifacts but **debugging** them, which is much more challenging, and thus the usability principle to "help users recognize, diagnose, and recover from errors" may be the greatest limiting factor in matching the authoring tool and task to the skill set of intended users.

Drawing insights from existing theories:

- Complexity Science provided principles for characterizing systems, artifacts, and procedures for using them, but says little about measuring complexity from the human and cognitive perspectives. For this we first bring in **Activity Theory**, which provides a robust vocabulary and theoretical framework for coordinating tools, tasks, users, and user communities. We can layer our focus on complexity on top of this framework, and speak about coordinating the complexity of tools, tasks, users (mental models and skills), and the communities of practice that support users. This provides an opportunity to speak not only to authors as individuals, but as members of knowledge building, design, or practice communities.
- Focusing in on the user, we look at approaches for specifying the level of "**cognitive complexity**" in terms of the complexity of the **mental model** that the user needs to understand or construct in order to use a tool. A useful framework for doing this is **Epistemic Forms** (and Epistemic Games), which allows us to classify features (interface components and procedures) into epistemic classes. These classes can roughly be ordered in terms of complexity. For example, filling in a text form or setting a slider level are examples of simple tasks, while filling in a table or hierarchy have medium complexity, and creating and debugging formal procedures has much complexity.
- The complexity of both the user/cognitive factors (mental models and task procedures) and various dimensions of system/tool properties could be measured **quantitatively** through systematic analysis of all of the parts, relationships, etc. (as indicated from Complexity Theory). One of the challenges here is the "**dimensionality issue**" of how to combine what we determine about the complexity of each dimension into a holistic impression of overall complexity. For example, we might qualitatively analyze the complexity of the number of components, but how to we combine that with the number of types of components, and with the level of dynamic complexity—is it possible to "compare apples with oranges," so to speak, in a holistic assessment?
- We suggest that the effortful detail of quantitative complexity analysis is not necessary for our goals (though it may be useful for some applications), and that a more **qualitative** and comparative approach will suffice. That is, we can categorize system complexity and cognitive capacity qualitatively roughly into types or groupings that will provide sufficient insight into the design questions about matching the complexity of tools to the complexity capacity of users. We began doing this with the categories of epistemic forms above, which partially address the

dimensionality issue because they provide a more holistic assessment of the structure of a system.
- We can thus construct rough **categories of low, medium, and high complexity** for matching tools to users. When more detail or specificity is needed, the frameworks mentioned in this paper can be employed to place tools and users into more precise categories.
- Using Epistemic Forms only partially solves the dimensionality (apples vs. oranges) challenge of integrating various types of complexity into a holistic picture. To further address this I draw from theories of **Hierarchical Complexity** (including Skill Theory, from neo-Piagetian developmental research). Hierarchical complexity theories give research-based support for the idea of content-general and holistic methods of specifying cognitive and task complexity (we use them here quantitatively but are also applicable for precise quantitative metrics of complexity). I use the principles of Complexity Theory to suggest a complexity ordering from simple objects, to abstractions/mappings, to formal systems, to dynamic systems, to systems of dynamic systems (architectures and ecosystems) that gives a theoretical basis for ordering epistemic forms (and thus mental models and tasks) in terms of complexity.

In the end we have (1) a rough model showing one example of what a mapping from authoring system features to user characteristics (complexity capacity) might look like—which describes low, medium and high complexity levels; and (2) a framework for creating more valid, tailored, or detailed models that accomplish the same goal. Such models can help authoring tool designers (A) assess the complexity capacity of intended author tool users, (B) weigh the risk involved in investing various levels of effort in authoring tool design; and (C) make design tradeoffs in usability and power such that tools are appropriate to realistic use contexts.

The (MY) Story So Far

Because authors for this special issue were invited to include some personal retrospective narrative, I will describe how it was that I ended up working on ITS authoring tools in the first place. After graduating from college with an undergraduate degree in Physics (and a minor in Philosophy) I worked in industry in an "advanced prototyping" R&D unit in a leading semiconductor manufacturer. At the end of three years I found the corporate life to be unpalatable and I bid it goodbye for a time. After a year of soul searching and travel, in 1983, I found myself at UMass in a radical "create your own degree" graduate program called the Future Studies Program (a department in the School of Education that had close links to Buckminster Fuller and other luminaries). The doctoral program I wanted to create involved a study of Physics and Philosophy, with a goal of becoming a writer who would further enlighten the masses about the deep themes I was excited about in popular books such as The Dancing Wu Li Masters and The Tao of Physics. Having had some experience with the early days of computing (in the punch-card days), I also took up with faculty and students in the Instructional Technology program, which had ever-closer links with Future Studies because the future was looking very digital. Looking back, I never really had the skills to

accomplish my original goal, and was therefor quite lucky to have been drawn to the courses on Intelligent and Adaptive Tutoring and Learning Systems taught by Beverly Woolf. Before long this was my new academic focus and I was spending more time in the Computer Science Department than in the Future Studies department. I dove into classes in Artificial Intelligence, Cognitive/Learning Science, and constructivist educational theories, in addition to slogging through course requirements for a degree in Computer Science.

The penchant for taking an ever wider system's perspective that drew me to Physics and Philosophy eventually caught up with me in my work in instructional systems. After some time doing research assistant work on a couple of ed-tech projects I began to form the plan for my dissertation research.[4] Woolf's dissertation work, and, at the point, her main scholarly "claim to fame" was a formalism for representing tutoring strategies called Discourse Action Transition Networks. My studies in Instructional Design Theory, Learning Theory, and my work with Clement had introduced me to the wide variety and complexity of strategies that were being proposed for teaching/learning specific types of knowledge (facts, concepts, principles, skills, analogies, mental models, problem solving, scientific inquiry, etc.—and numerous sub-classes of all of these). Woolf's formal DACTN framework (Woolf and McDonald 1984) had only been implemented in prototype domains, and I was interested in what it would take to represent multiple strategies and meta-strategies in an intelligent tutor. I wanted the research to be general, grounded, and extensible by working with real instructors in several domains. I wanted the system to be usable—to have real instructors be able to create, modify, or at least inspect and select, among a library of theoretically grounded teaching strategies.

As is often the case for dissertation plans, the scope of my work was reduced by half, and half again, before I completed. I realized that the first thing I needed to do to be able to work flexibly with a wide range of tutoring strategies and domains was to build an authoring tool so that strategies and domain knowledge could be visually represented clearly enough for domain experts and teachers to understand them; and for the researcher to easily modify and experiment with alternative strategies. Creating a domain-independent ITS *authoring tool* was to be the first step; researching and representing a wide variety of strategies was to be the second; and working with instructors to represent domain knowledge and test and improve the teaching strategies was the final planned step. Looking back of course, the plan was amusingly over-ambitious, as dissertation proposals can be. The task of just creating the authoring tool occupied the entire project, and continued to be extended in grant-funded research for many years (with the KAFITS and Eon systems).[5] Results of Encoding Knowledge with Tutor Construction Tools. In the Proc. of the Tenth National Conference on Artificial Intelligence (AAAI-92). San Jose, July 1992, pp. 17–23.). We did represent a number of domains and teaching strategies in our ITS authoring tools, but the vision of having an overarching system for knowledge types and teaching strategies was elusive (and, I now know, may always be elusive, though the team working on the

[4] My first major research project was with Clement, who was a senior member in a novel program called the Scientific Reasoning Research Institute housed in the Physics department. The department was known as a hotbed of radical constructivists, including Ernst Von Glasersfeld, and sponsored leading-edge research and applied programs in constructivist approaches to math and science.

[5] KAFITS: Murray and Woolf 1992; Murray 1996; Eon: Murray 2003b and www.tommurray.us/eon_www/eon.html.

GIFT authoring system has come the furthest thus far; see Kumar et al. 2010; Sottilare et al. 2012).

The topic of authoring tools within the ITS community was relatively new and fast expanding. I am the kind of person who hates the thought of "reinventing the wheel" or of missing some important relevant corner of the R&D literature. As I continued my work I kept up an interdisciplinary overview of related work. It seems that I was the only researcher at the time motivated to write a full overview of the state of the art. From my later studies in "Hierarchical Complexity Theory" (Commons and Richards 1984; Commons et al. 1998) I now know that what I accomplished was a "meta-systematic" or perhaps "paradigmatic" overview the field. That is, in addition to describing the authoring system projects and theoretical frameworks in existence, I organized and coordinated these perspectives into a coherent whole, and, so to speak, organized perspectives on perspectives in my overview publication(s). One could say that the design of authoring tools is "ITS-complete" (a play on the term NP-complete), in that to design a generic software tool one needs to create a generic representational framework that anticipates both practical and theoretical issues. Therefor, a full treatment on authoring tools will tend to include a type of overview of ITS theory in general. My 1999 Overview paper included sections on:

- Types of ITSs and domains that have been authored—a categorization of 29 existing ITS authoring tools into 7 primary categories for analysis;[6]
- *Methods for authoring* the Interface, Domain, Teaching, and Student Models
- General authoring/*knowledge acquisition* methods
- *Design tradeoffs*—including Power/flexibility (=Breadth x Depth); Usability (=Learnability x Productivity); Fidelity; and Cost. These tradeoffs exists for all ITS components individually (domain, tutoring, and student models; and learning environment).
- An outline of 5 ITS *authoring roles* and their related skill-sets, with an analysis of what types tool features should exist for each role (similar to Table 1 shown later).
- Case studies and *pragmatics* (actual use scenarios and descriptive statistics, efficiency estimations, evaluations performed)

Though authoring tool R&D has progressed, the categories and frameworks I suggested seem just as relevant today as then, which may explain why the paper is still cited.

The focus of my work has moved away from research per-se on ITS authoring, but I have continued to build special-purpose authoring tools for almost every computer-mediated learning project I have been involved with (MetaLinks, Rashi, SETS, Wayang, and Simforest-G—see descriptions at www.tommurray.us). Though these projects did not aim for as much generalizability as the Eon or KAFITS frameworks, I continued to face and learn from the issues highlighted in the earlier work.

[6] The paper described the Strengths, Limits, and Variations among ITSs with these areas of strength or specialization: Curriculum Sequencing and Planning, Tutoring Strategies, Device Simulation and Equipment Training, Domain Expert System, Multiple Knowledge Types, Special Purpose, Intelligent/Adaptive Hypermedia.

Table 1 Authoring tool user roles and complexity capacity estimates

Roles (tool use roles)	Benefits (of that role)	Problems (of that role)	Complexity Capacity for ITS design
Teachers PRACTICAL	Practical experience	Not good at articulating or abstracting expertise	LOW
Domain Experts & Content Developers PARTIAL	Auth. tool infers the instructional methods	A fixed instructional method	MED
Instructional Designers & Learning Theorists THEORETICAL	Know learning theories & research	Rare; not trained in knowledge engineering	MED
Knowledge Engineers and ITS Developers EXPERIENCED	Know the tools; Are sometimes also plugged into user testing	May not know what it is like to teach or learn the material	MED-HIGH
Computer scientists & Software developers (ACTUAL?!)	Complexity capacity. Do not have to build to a real user base.	"its intuitively obvious to the casual observer…"	HIGH

Challenges Facing Authoring Tool Research Today

Predicting Future Flying Machines ITS authoring tool research is in an interesting socio-techno–historical position. Intelligent Tutors, despite 30 years of R&D, are not yet common in mainstream education or training, though a few notable systems have achieved wide-spread use (Koedinger et al. 1997; Heffernan and Heffernan 2014; Graesser et al. 2005; Vanlehn et al. 2005; Mitrovic 2012; Johnson and Valente 2008; Sitaram and Mostow 2012). This may be a completely appropriate development and adoption arc for a technology this complex and innovative, and we have every reason to believe that the results of advanced technology learning systems (ATLS) research will continue to influence on-the-ground computer-mediated learning. However, authoring tool researchers are in the awkward position of developing the cart before the horse. Or, worse yet, developing the cart-factory before the horse. It is as if, as the Wright brothers were experimenting with the first airplanes, a group of researchers and academics were observing on the side, working out how to design airplane factories that would make airplane production efficient and flexible. As those first manned flight contraptions were being developed, it would have been difficult to predict what future flying machines would look like, never mind what the market would be like or how to best mass-produce and easily customize them for typical users.

Of course, ITS work is well beyond its first prototypes, so this analogy is stretched. Still, authoring tool designers work under considerable uncertainty as to what types of systems will find their way to substantial use and benefit from the scale and flexibility that authoring tools enable. However, we are talking about software here, not equipment manufacturing. Building abstractions and design tools is a natural impulse in software design (procedural-, data-, and knowledge-abstraction are basic computer science principles—see Abelson and Sussman 1983). As indicated in the history of my own projects, it can be beneficial to build authoring tools merely to facilitate local or small scale R&D projects. A company that makes a decent profit on one single piece of widely used software (say an ITS) would benefit from building authoring tools to customize and enter content for the ITS. However, for systems built for in-house use it is more difficult to frame scholarly research questions and findings.

Old vs. New Conceptions of What an ITS is The original understanding of computational "intelligence" in ITS's involved mostly modeling and knowledge representation tasks—learner, domain, and instructional models. The more deeply Cognitive Science understands knowledge and learning the more difficult these modeling tasks appear for authentic situated tasks. In general the most successful ITS's are those focusing on knowledge that is the easiest to represent, including declarative facts and procedural steps. Yet developments in Learning Theory increasingly emphasize the importance of forms of knowledge that are more difficult to formalize, such as metacognition, conceptual understanding, problems solving, open ended inquiry, collaboration, communication, argumentation, hypothetical and analogical thinking, etc.

The more basic forms of knowledge (fact, skills, and concept-map-like relationships) continue to have fundamental importance as building blocks for more sophisticated skills, but the more exciting work in ITS/ATLS has been moving into a wide variety of areas that do not involve "deep modeling" of knowledge or expertise. These new research trends include: recognizing and responding to affect; using big data to classify

and predict learner behavior, wearable gadgets, immersive experiences, natural language understanding and production, game-ification and social-media-ification. For a project to be considered "ITS" research it no longer requires computational intelligence per se, but only the inclusion of some state-of-the-art computational technology (or leading edge techno-socio-psycho theory). While the idea of a generic ITS framework requires some commonality of basic components and/or representational frameworks, the scope of ITS's is becoming increasingly diverse, and overarching frameworks are increasingly difficult to envision. However, once could counter that as diversity increases, so does the number of projects, so that the actual impact of designing generic frameworks still serves a significant (if smaller percentage-wise) potential user base.

Authoring tools are still essential for scale-up, wide adoption, and easy customization of learning systems, though each may need to be specific to a very specific genre of instructional systems. If so, authoring tool design may become more of an engineering challenge than a research area. However, there are still important theoretical issues that can be investigated, which we explore next.

Software Usability and Complexity

Usability and Managing Software Development Risks Bracketing the above concerns, let us assume that ITSs of some sort will indeed become mainstream and that authoring tools will become increasingly important—a safe bet I think. Other than tools designed for in-house use by highly trained specialists, authoring tools, by their nature, must be usable by some anticipated user audience. As mentioned, with any tool there are *context-specific* usability concerns that can be worked out through good design and HCI practices (prototyping, early feedback from authentic users, etc.), but here I would like to look at very *general* usability concerns, having to do with the complexity of these systems.

ITSs are complex software applications and full-featured ITS authoring *tools* can be an *order of magnitude* larger and more complex—just as a machine designed to build many types of lamps is much more complex than a lamp (though the machine itself may be relatively easy for the end user/author to use, its interiors will be more complex). Next we will look to the literature on the *design and usability of complex software systems* for advice relevant to ITS authoring tool design.

Design tasks such as authoring ITSs fall under the "ill-defined" and "wicked" problems characteristic of real-world projects (Conklin 2005; Mirel 2004). In his treatment of usability of complex systems Oja (2010) defines complex software development in terms of Mirel's definition of complex problem-solving, which involves "ill-defined situations; vague or broad goals; large volumes of data from many sources...; nonlinear, often uncharted analytical paths; no pre-set entry or stopping points; many contending legitimate options; collaborators with different priorities; [and] 'good enough' solutions with no one right answer." Chilana et al. (2010) give three additional factors that contribute to the complexity of designing usable software: domain-specific terminology, every situation is unique, and limited access to domain experts. ITS/ATLSs and their authoring tools certainly have all these characteristics.

Oja contends that Nielson's classic usability heuristics are even more critical for *complex* software development (Nielsen 1994). Nielson's usability heuristics include:

reification (visualization of key abstractions and relationships; minimize working memory load), user control and freedom (not constraining user actions any more than is necessary), flexibility in outcomes (to allow for variations in style and needs), match between system and the real world (using the vocabulary and mental models users already have); help users recognize, diagnose, and recover from errors, user control and freedom and flexibility and efficiency of use.

Echoing the heuristic to "match between system and the real world," Johnson (2006) analyzed software usability failures in the Healthcare sector that imposed significant financial and acceptance burdens within that sector, and found that "many usability problems stem from the inability of suppliers and manufacturers to anticipate [user] requirements." The educational technology R&D community is poised to create ITS authoring tools that could be used on a large scale. As the investment in authoring tools increases, there is a corresponding increased "risk" that investment in design, outreach, etc., will outweigh the benefits if the tools do not directly meet the needs of a wide variety of users (or if the ITS that is built with the tools does not reach a large number of learners).

Figure 1 illustrates the type of risk management and risk reduction principles increasingly being used in software and other industries.[7] Additional investments in software can follow the "80/20" rule where perfecting the last 10 % or 20 % can take a disproportionate amount of effort. Meanwhile, the return on user value gets proportionately less. The goal is to find the sweet spot where risk is acceptably low and expected value is relatively high ("optimum" in the Figure). To mitigate this risk usability principles recommend both empirical and theoretical grounding: i.e. usability evaluation and user-feedback from authentic contexts done "early and often;" and a good theoretical understanding of the user and task. Complexity is a useful construct for operationalizing Johnson's "[ability] of suppliers and manufacturers to anticipate [user] requirements," but the construct needs better definition for this to happen—which is what we hope to contribute to here.

Complexity Science and Information Theory Next we branch away from complexity in software and usability theory to consider how complexity is theorized in more general terms. Complexity Science points to various methods for measuring complexity which are all related to the amount of information contained in an object, system, or process, with "information" being closely related to the concepts of difference, discernibility, and degrees of freedom. Information and communication theories also quantify information (and "meaning") in terms of entropy, randomness, chaos, "surprise," and "shortest possible description" (Grünwald and Vitányi 2003). There are many individual metrics that contribute to overall complexity, including the number and diversity of components and their structural or functional relationships (Benbya and McKelvey 2006). Complexity Science also deals with time-based phenomena: change, feedback loops, self-organization, evolution, and emergence in dynamic systems—so-called "complex adaptive systems."

[7] Image adapted from http://www.labcompliance.com/tutorial/risk/default.aspx?sm=d_a, "Risk Management in the (Bio)Pharmaceutical and Device Industry," L Huber & Labcompliance Inc., http://www.labcompliance.com/tutorial/risk/default.aspx?sm=d_a.

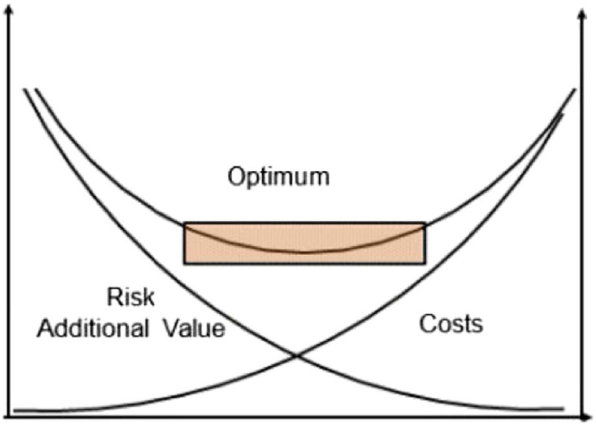

Fig. 1 Cost vs. value in software risk assessment

Campbell (1988) describes three sources of complexity: number of *dimensions* of information, the *rate* of information change, and the number of *alternatives* associated with each dimension (i.e., information diversity). We will modify and generalize this scheme as in Fig. 2, using the categories of structural, dynamic, and perspectival complexity.

For structural complexity, other things being equal, systems are more complex if they have: more parts (e.g. an ant colony or huge Lego project); more types of parts (e.g. a car or human anatomy); more properties in each part; more relationships or constraints among the components; and more types of relationships. In part relationships, one-to-one mappings (relationships) are the simplest, one-to-many mappings are more difficult, and many-to-many mappings are most complex to manage and conceptualize.

In addition to these structural dimensions (which are metaphorically space-like), systems whose properties, relationships, and objects *change* over time are more complex (the dynamic or temporal dimension). Dynamic complexity is represented

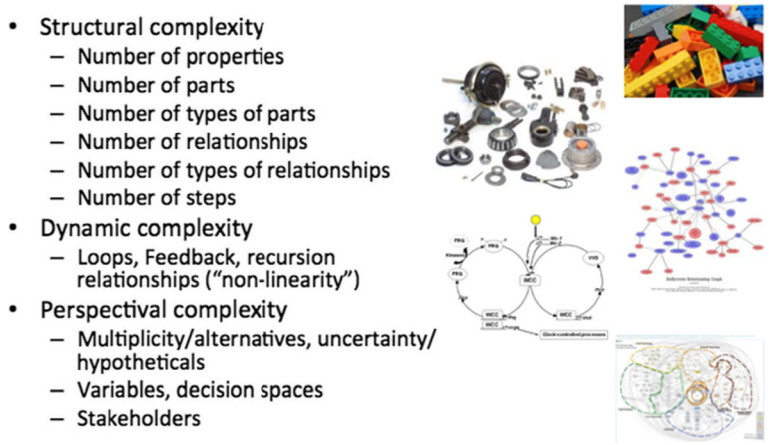

Fig. 2 Sources of system complexity

using laws, rules, mechanisms, or influences. Feedback loops and nonlinear dynamics, all outside our scope to elaborate on, come into play.

As indicated above, complexity is related to information intricacy, space of possibility, and even "meaning," and is thus not simply an objective property of systems, but it has a quasi-subjective component that involves human context, activity, and the reasons for doing the complexity analysis. In software, information systems, and usability analysis there are cognitive and epistemic considerations. Byström & Järvelin's analysis of task complexity includes factors such as: repetitiveness, analyzability, a-priori determinability, number of alternative paths, outcome novelty, number of goals and conflicting dependencies, uncertainties between performance and goals, number of inputs, and time-varying conditions of task performance (Byström and Järvelin 1995, p. 5). Zhang et al.'s (2009) "epistemic complexity" measures complexity in terms of the movement from facts to explanations and from unelaborated to elaborated knowledge—both of which indicate increasing depth and complexity. Epistemic complexity includes measurement of the "diversity" and "messiness" one encounters in a situation (Bereiter and Scardamalia 2006). Thus concepts of nuance/subtlety, abstraction/generalization, uncertainty/ambiguity must be considered.

Therefore, in Fig. 2 we have the third category "perspectival" complexity, which is complexity due to multiplicity and uncertainty, including conflicting goals or subtasks; diverse perspectives among stakeholders; stochastic randomness and indeterminacy; and vagueness and uncertainty in any of the structural or dynamic elements (measuring these would be more heuristic than the other two complexity factor types). Perspectival factors relate as much to subjectivity and the nature of cognition as to the objective nature of the artifact.

Usability Complexity and Runnable Artifacts In terms of software systems, specifically authoring tools, the factors mentioned above can be applied to the software artifacts (code and interface), development (programming or authoring) or the complexity of use (the user interface understanding and the mental model a user must acquire to understand a system). Theoretically each of the sources of complexity in Fig. 2 could be enumerated or estimated and combined to measure the complexity of a system toward the goal of comparative analysis of the complexity of systems.

Artifacts that 'run' or behave dynamically are of course more difficult to author. With authoring tools and educational software such as Scratch and StarLogo, and scripting languages in Office applications, the line between programming and using software is increasingly blurred. ITS authoring can fall anywhere along a spectrum of complexity from customizing parameters and choosing content to creating teaching strategies, which is closer to software programming.

ITSs are dynamic systems that must be run to test them. They have multiple learning paths and it is intractable to test every possible student behavior. Unpredictable behaviors inevitably occur in complex software. The simplest systems have predicable paths with little interaction or parameterization, such as Scripts and story-board type procedural flows. If an authoring tool allows branches, if/then rules, procedures, loops, parameterized subroutines, or recursion (in rough order of difficulty) the level of authoring complexity jumps dramatically. The author is essentially doing software programming. Writing and debugging computer programs is a complex task requiring

special skill and tools. Without these skills, and even with them, it can be quite difficult to determine the source of a run-time software bug.

Creators of authoring tools that allow authors to enter into this level of task complexity must (1) not underestimate the complexity of the task or overestimate the skill of the typical user, and (2) provide real debugging and tracing tools—for the systems to be viable. One of Nielsen (1994) "Top 10″ recommendation for usability is to "help users [authors in this case] recognize, diagnose, and recover from errors." This can be as simple as providing an Undo feature for authored content, but for systems with dynamic complexity special tools are needed to trace and debug procedural representations.

Like most software systems, ITSs should be designed in user-participatory feedback loops, where, as Benbya & McKelvey note, "the critical factor in all information systems is continual change" (from "Toward a complexity theory of information systems development", 2006, p. 20). This might even imply that viable authoring tools should have some sort of "version control" subsystem.

The above discussion suggests factors that could be considered in characterizing the complexity of software tasks and interfaces. It is implied that for some tasks, such as version control and debugging, there is a need for special skill such as knowledge engineering. Thus it is also important to consider the "complexity capacity" of users and communities of practice—and for this we turn next to Activity Theory.

Activity Theory—Users, Tasks, Tools, and Communities

We borrow concepts from Activity Theory, set of principles about socio-technical human action and design, which stresses the mediating role of *tools* (artifacts) and their usage *rules* in collective human activity and development (Jonassen and Rohrer-Murphy 1999; Stahl 2006; Engestrom et al. 1999). Here *rules* indicate the (sometimes implicit) skills, understandings, and habits held by a community of practice. Thus we can frame our exploration of authoring tool usability in terms of the interaction between *users*, *tools*, *rules*, and *tasks*. We can ask whether a tool and its "rules" of use afford the accomplishment of a particular task for a particular class of users. Clearly our *users* are authoring *tool* users and the *task* is to design or customize an ITS; and later we will introduce "epistemic forms/games" as a way to describe the *rules* of use.

Figure 3 illustrates these factors in Activity Theory terms (adapted from Jonassen and Rohrer-Murphy 1999; Engestrom et al. 1999). Thus, from our focus on the concept of complexity, we must consider:

- Task and Rule complexity (user activity methods and goals)
- Tool (artifact) complexity
- Socio-cognitive complexity (community of practice and division of labor)

We are concerned with the match between:

- User vs. Tool complexity
- Task vs. User complexity
- Community of Practice vs. Tool complexity

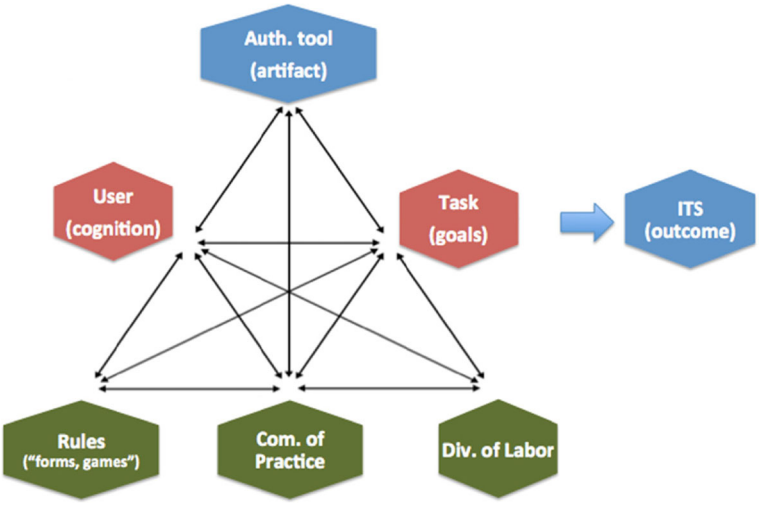

Fig. 3 Activity theory

When we speak of users we are really speaking of users in particular *roles*. This distinction is important when we begin to speak of the complexity capacity of a user (or type of user)—we are not referring to a person's general ability to handle complexity, but to one's ability within a certain role (ITS author, content developer, tester, etc.), which might depend more on training and experience than on innate intellectual sophistication).

Campbell notes that there are several approaches to assessing complexity: as a subjective psychological experience of the user, as an objective measure of the task, and as an *interaction* between subjective and objective elements (1988, pg. 44). While measuring complexity in terms of user (author) *experience* is important, methods for doing so are outside our scope here. However, we will describe methods for describing user *capacity*, and we assume that on average complexity capacity is closely related to the complexity experience of the user (they will be frustrated or confused if their complexity capacity in a particular role is mismatched for the task). In the prior section we sketched heuristic frameworks for assessing task and tool complexity objectively. Our eventual goal is to assess the match (or interaction) between user capacities and the measures of tool/task complexity (user capacities will be roughly estimated, while tool/task complexity affords more objective measurement).

Note that in the prior section tool and task complexity were treated together. Unlike simple tools such as a hammer, for which the task a tool is used for (e.g. building a barn) is usually much more complex than the tool itself, for most software tools the complexity of the tool features can stand as a fair indication of the complexity of the task. This is of course not strictly true, as building an ITS involves much more than using an authoring tool (e.g. applying learning theory, paper mock-up design, etc.), but for simplicity we will assume that the complexity analysis given above of artifacts (tools) maps well to complexity analysis of tasks. Task-related issues of *how* the tool is used and learned will be categorized in Rules or COP (community of practice) elements of Activity Theory, rather than with the artifact.

Epistemic Complexity and Complexity Capacity Oja quotes Haynes and Kannampallil (2004) who say that "complex software applications require great cognitive skill, integration of knowledge from various areas, and advanced instruction and learning; thus, it is not surprising that 'screen deep' interfaces to such systems may not yield the best results in terms of usability." This is one reason why understanding the intended user is so important—because making a tool more easy to use, i.e. "usable," may dumb it down too much for some users or tasks, and decrease "user control and freedom" and "flexibility and efficiency of use" (from Nielson's model) for those contexts. Oja (2010): "As Mirel (2004) points out, most current HCI practices concentrate on ease of use or simplifying the work, and this may lead to 'producing good designs but for the wrong problems'"(p. 3800). The design goal is thus to make tools "operationally simple, while intellectually sophisticated and nuanced" (ibid).

"Cognitive complexity" is one term used to describe a person's capacity to perform complex mental or behavioral tasks. Cognitive complexity involves not only the number and complexity of the objects and relationships as described above, but also the ability to perceive nuances and subtle differences—i.e. it can involve both integrative and differentiating capacities (Mirel 2004). Jordan uses the term "complexity awareness" for "a person's propensity to notice...that phenomena are compounded and variable, depend on varying conditions, are results of causal processes that may be...multivariate and systemic, and are embedded in processes [that involve non-simple information feedback loops]" (Jordan et al. 2013, p. 41). As mentioned above, Zhang et al. (2009) use the term "epistemic complexity" which includes an understanding of underlying reasons, theoretical explanations, or hidden mechanisms within phenomena. Below will use the term "complexity capacity" to remind us that cognitive complexity required for a task is about the context and role a person is in, and depends on experience in addition to any general complexity "intelligence" they may have.[8]

In our exploratory discussion of software usability and complexity we enumerated many factors and it remains for future work to determine how these factors are operationalized, weighted, and combined in any overall complexity metric (a process that may be quite context-specific, as complexity components will have different weights for different situations). As we move from characterizing the complexity of tools (software) and tasks (authoring) to that of users, the approach will continue to be preliminary and suggestive, with many details remaining to be worked out beyond this paper. Lets assume, for simplicity, that we have worked out the details of a scheme such as the one described in prior sections of this paper, and have devised a method to characterize *task*/*tool* complexity level, and that we have collapsed the dimensionality of analysis to rate tasks/tools on a scale of low...medium...high complexity. How might we map this to *user* (or community of practice) complexity capacity? Table 1 illustrates what such a mapping *might* look like, showing types of authors, benefits and problems typical of each author type, and the level of design complexity one can typically expect in the authoring task.

Teachers have on-the-ground experience of the needs of students and classroom situations, and, while their input should be included in the iterative design process, they

[8] Cognitive complexity might be used as an informal construct useful to differentiate different potential users, but it may also be possible to measure it in a way relevant to ITS authoring. A Google Scholar search of "measuring cognitive complexity" yields pointers to many attempts to do so.

cannot be expected to have the skill, nor the time, to use or learn how to use complex authoring tools. Domain experts and content developers are more typically used to define knowledge and expertise, though they may have little practical or theoretical knowledge of pedagogy. Instructional designers and learning theorists bring different sources of pedagogical knowledge and epistemological knowledge (understanding how knowledge is structured), though they may not have the time to dedicate to a steep tool learning curve.

For all of the above user types, the task of representing knowledge in a computationally usable fashion may be foreign—while *knowledge engineers* are trained in exactly that task. It is only with this level of skill and higher that we can expect sophisticated authoring tasks to be managed.[9] Most user communities will not have people with knowledge engineering (or ITS design) skills, meaning that users at this level will usually be part of a dedicated ITS design team, which would only exist in an academic lab, a company dedicated to building learning systems, or an educational organization large enough to form such a team to be shared widely (e.g. a university or city school district).[10]

The final category of users in Table 1 is computer scientists and software developers. This category connotes the unfortunate yet understandable fact that many ITS authoring tools never see a robust user community and are only used within the confines of the team or organization that built the tool. This stakeholder group tends to be the most sophisticated in terms of designing complex structural and procedural models. The benefit is that more powerful ITSs can be built, but the drawback is that without usability input from "real" users, the tools may be too complex to expect many others to pick up, and the tool designers may be out of touch with the needs of intended users.

In authentic contexts the actual "capacity" of a user to use a tool to accomplish a task depends on "community of practice" considerations as well as the potential complexity capacity level of the individual (Fig. 3). These considerations include: (1) opportunities, investment and incentives in *training*; (2) community of practice peer and mentor *support*; and (3) *time* available to author. Thus, even if a user, say an unusual teacher, has a high level of generic complexity capacity, in order to successfully make use of an ITS authoring tool they would need to be able to invest time in the learning curve, have the support of peers and superiors in adopting this new technology, and have the ongoing time available to do the authoring. Contexts satisfying these conditions are indeed rare. It should also be noted that actually it is the capacity and skill set of the design/authoring *team*, not of any individual, that is important.

In addition, for newly introduced artifacts there is a dynamic, often evolutionary, interplay between artifacts (their design), the standard and novel ways that artifacts are put to use, and the human capacities enabled by artifacts. That is, new tools create new capacities, which create new possibilities and new goals/tasks; around which new (or improved) communities of practice develop—all of which in turn prompt new

[9] Here "sophisticated" refers to the complexity of the authoring task, not the domain knowledge of, say a teacher, which may be sophisticated in another way.

[10] Note that this specific scheme is suggestive and meant to illustrate a framework rather than the "content" of the framework—i.e. I do not need to make a strong argument here that, e.g. "Domain Experts & Content Developers" have a limited or "fixed" understanding of instructional methods, as is given in the Table. Of course, the roles in the Table can be combined in any individual, but it would be rare that, for example, a classroom instructor would also be a learning theorist or knowledge engineer.

innovations (tools) to continue the cycle. Benbya & McKelvey (2006, p. 14) refer to the "co-evolutionary" aspects and "adaptive tension" of the "complex adaptive" socio-technological systems, and discuss the problem of "accumulating requirements". So, an important community-of-practice question is: How effective are the feedback and *development learning loops* between users, trainers, and designers?

Thus far we have described what a tool/task/user complexity mapping scheme might look like, without saying much about the nature of user cognitive complexity. A user's understanding of tools, tasks, and methods can be described in terms of the *mental models* one has of these things (Gentner and Stevens 1983; Johnson-Laird 1983). Mental models are cognitive representations of external systems that include structures and processes that a person simulates (runs or visualizes) mentally. One task of the authoring tool is to help the user maintain a valid mental model of the ITS building blocks, range of configurations, and design steps that the authoring tool affords.

Oja notes that "Cognitive engineering (Gersh et al. 2005) and learner-centered design (Soloway et al. 1994) focus on improving system-human cognitive fit and allowing users to construct better mental models (knowledge) of the system" (p. 3801), and that "reification is the basis for successful communication and the establishment of a shared goal in human-computer collaboration" (p. 3803). Thus it is important that the authoring tool interface accurately and powerfully reify the structures, objects, constraints, decision rules, and procedures involved in authoring, so that authors can build correct mental models, and can use these mental models to coordinate the various steps and roles within a design process. The complexity of mental model that is supported in the authoring tool should match the complexity capacity of the user.

Collins and Ferguson's work on "epistemic forms" provides a valuable link between task/tool complexity and the user's complexity capacity in terms of the mental models that the user must construct and maintain. In its concept of "epistemic games" it also anticipates the community-of-practice element of Activity Theory. We discuss epistemic forms and games next.

Epistemic Forms and Games

Collins and Ferguson first articulated the concepts of epistemic games and epistemic forms (Collins and Ferguson 1993; and Morrison and Collins 1995; Shaffer 2006). Epistemic *forms* are "target structures, like mental models, that guide inquiry" and are "recurring forms that are found among theories in science and history." Epistemic *games* are "general purpose strategies for analysing phenomena in order to fill out a particular epistemic form" that are shared within a community of practice (p. 25). Example epistemic forms include lists, hierarchy or tree structures, tables, networks, if-then rules, and constraint-based systems. They are "generative frameworks with slots and constraints on filling in those slots," and in this sense are like domain-independent scripts, templates, or grammars that specify the structural properties of a phenomena. They serve as commonly understood mental models for understanding tasks and tools.

The theory of Epistemic Forms/Games considers not only the structure of information, but also the ways (i.e. games) communities use, understand, and build knowledge using that structure. For example, perhaps the simplest epistemic form is the list. Knowing how to play an epistemic game includes knowing its constraints, strategies,

and moves. For the "list game" this includes knowing how to add, remove, combine, split, and arrange (classify, filter or sort) items, and knowing when the "list form" is most appropriate for a particular problem or inquiry. This framing is compatible with Activity Theory, which highlights the interplay between cognition, artifact design, and communities of practice.

Morrison and Collins coined the term "epistemic fluency" to refer to the ability to use and choose appropriately among the repertoire or ecology of epistemic games available within a community of practice. Epistemic games are rarely used in isolation, and are combined with other games as well as transformed into other games, as when one representation (a concept network) is seen as more appropriate than another (a table). Tables can be seen as composed of lists; even more complex forms might combine tables with networks (e.g. a network of tables, or a table of networks). Table 2 lists some Epistemic Forms/Games mentioned by Collins & Ferguson.

Epistemic games can be framed in terms of the key questions driving an inquiry. Knowing an epistemic game includes knowing how to evaluate whether it is being played well. Example quality/validity criteria for the list game include coverage (is anything missing?), similarity (do the items belong together, or should it be split into two lists—apples and oranges?), distinctness (are the items actually different?), and perspicuity (is it sufficiently short, simple, efficient, and understandable?). Vibrant communities of practice will be creating, tweaking, and evolving, and mashing up their epistemic games.

Authoring Tool Epistemic Forms Epistemic forms/games allow for a compact method of classifying tool/task complexity. In our original discussion of artifact complexity we suggested that one could enumerate the number and types of parts, properties, relationships, etc. in a system. This may be useful to do but also quite cumbersome. Meanwhile, epistemic forms serve well as a first-pass description of the complexity of end-user software systems. Epistemic forms also address one difficult issue in the characterization of an artifact, which we will call the "dimension compression problem": it may not be difficult to classify and compare artifacts along any single dimension (in Fig. 2), but we have little guidance thus far on how to combine and prioritize the many dimensions into a single (or simple) complexity characterization. Epistemic forms are holistic and representationally efficient in that they incorporate many of these dimensions into each category.

Table 2 Epistemic forms and games (mental models) (from Collins and Ferguson 1993)

• list	• street map
• matrix or table	• org. chart
• molecular model	• musical score
• periodic table	• timeline
• web page menu	• cause/effect diagram
• x-y graph	• network
• pert chart	• relational database
• binary tree	• sentence diagram
• floor plan	• term paper outline

In discussing authoring tools we are interested specifically in design activities or *design games* (a term not used by Collins and colleagues). In all epistemic games one of the evaluation criteria is whether one's product is understandable or meaningful to others *within* one's community, while *design* games are distinguished by the additional need to assess how understandable and useable the product will be to *users* (who belong to a community related to but different than the designer community). Thus, the set of design game quality/validity criteria is extended to a group that requires some cognitive empathy (and design/test iterations) to serve well.

In surveying a set of 14 authoring tools mentioned in Murray et al. (2003) one can clearly see a set of epistemic forms that are repeated numerous times throughout most of these systems. This list of forms will not be surprising—they are seen in most software tools, as shown in Fig. 4. The basic elements include: check boxes and choice lists; sliders, dials, and meters; graphical networks and trees; and interactive hierarchical and tabular textual representations. As discussed, to compare across and within any class of epistemic forms (say a hierarchical menu system) we can use the elements suggested in the earlier discussion of Complexity Science—i.e. the complexity of an interface and task includes the number and diversity of such elements and the degree of their inter-relationship or coupling in an overall system.

Intuitively one can roughly compare or rate the complexity of epistemic forms. Lists, sliders, and checkboxes are simpler than hierarchies, tables, and concept networks, which are in turn simpler than the complex systems/mental models that are composed of dynamic the interactions among many simple sub-components. *Hierarchical Complexity Theory* offers a more rigorous and more theory-based foundation for rating and comparing complexity components, and it was developed to apply to human tasks and skills. Next we explore HCT next as the last theoretical territory of exploration in our journey to link several interdisciplinary fields.

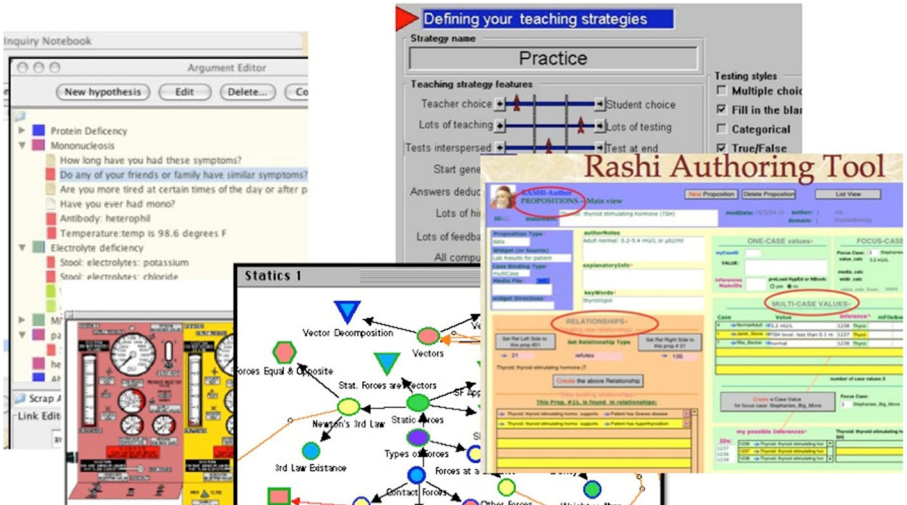

Fig. 4 Epistemic forms in authoring tools

Hierarchical Complexity and Skill/Task Development

Above we drew from information/systems theories and socio-technology theories (Activity Theory and usability theory) to suggest ways to characterize the complexity of systems in general terms. Epistemic Forms suggest a way of ameliorating the "dimensionality issue" by enumerating common forms that are more intuitive and ready-to-hand than a list of low level complexity dimensions. But we are still far from a quantitative or semi-quantitative method for combining the factors involved to be able to make comparative complexity judgments. To move in this direction I will draw from an area of cognitive/learning science that has significant implications for learning theory and ATLS design in general, yet, curiously, is rarely referenced in these fields: Neo-Piagetian developmental theories. Cognitive developmentalists (Neo-Piagetian theorists) have undertaken a deep study of complexity, because human development and learning can be described in terms of "qualitative differences in mental complexity" relative to various tasks, skills, or life contexts (Kegan 1994, p. 152).

Because these theories have untapped value for the ATLS/ITS community I include an extended description in the Appendix but here I will only mention the features of these theories that are relevant to our authoring tool complexity analysis. The key insight is that development, and complexity in general, advance through both horizontal and vertical ("hierarchical") movement, and do so through a particular alternating or spiraling pattern.[11] The structure and nature of horizontal growth is different than the structure and nature of vertical growth. Vertical growth is more quantized or punctuated, and the vertical leaps involve particular challenges. If we frame authoring tool features, authoring tasks, and epistemic games in terms of vertical and horizontal differences in complexity we have additional tools for comparing complexity, and we gain insight into why certain forms may be particularly difficult for users to learn.

Neo-Piagetian (adult) developmental theories go beyond early developmental work (E.g. Piaget, Perry, Kohlberg) to add a hierarchical "structural perspective in analyzing changes in the organization of …actions and thought" (Fischer and Yan 2002, p. 283). These theories propose underlying representations for skills and suggest rules for the transformation of skills to higher level skills.[12] These theories apply principles from Complexity Science to human cognition and behavior, which can be easily mapped onto artifacts (tools). As stated by Commons & Pekker: "Theories of difficulty have generally not addressed the hierarchical complexity of tasks. Within developmental psychology, notions of hierarchical complexity have come into being in the last 20 years. [...] a model of hierarchical complexity, which assigns an order of hierarchical complexity to every task regardless of domain, may help account for difficulty" (2009; p. 2).

Horizontal increases in complexity involve adding more of what already exists to an object, process, or structure (more parts, relationships, steps, etc.—adding more "bits"

[11] These models have been empirically demonstrated in many hundreds of studies. For example Commons and Pekker (2009) states: "Using [Rasch analysis we have] found that hierarchical complexity of a given task predicts task performance with the correlation being $r = 0.92$. [which] has been shown to account for performance in a variety of different domains" (p. 4).

[12] Fischer's Skill Theory (Fischer 1980; Fischer and Yan 2002) and other Neo-Piagetian models, including Commons' Hierarchical Complexity Model (Commons and Richards 1984, Commons and Pekker 2008), Kegan's stage model (1994, 1982); and Cook-Greuter's ego development model (2000, 2005).

of information without adding new structural emergence). Commons suggests that increases in the horizontal complexity of tasks (which he calls the "classical" model of information complexity) are analogous to increases in cognitive load (Commons and Pekker 2009 unpublished). Horizontal growth can also be roughly compared to Piaget's assimilation, as it adds new knowledge in the form of existing structures (Piaget 1972). Vertical growth related to accommodation, in which new structures are created to understand the world in new ways. Horizontal growth tends to be continuous, while vertical growth follows a more discrete model, and occurs after a sufficient amount of horizontal growth allows for a reorganization at the next higher level.

Vertical increases in complexity lead to a new level or stage by applying an operation upon, or "coordinating and transforming," the objects of the lower layer. Each artifact or skill at a given hierarchical level consolidates a set of items at the lower level into a single whole, transcending and yet including them. Completely new properties and concerns arise at each level (a phenomena called emergence). Examples of increasing levels of hierarchical complexity include the development (or evolution) from: words to sentences; addition to multiplication; single celled to multi-celled organisms; concrete to formal operational concepts; from using to designing an artifact; and from doing a task to managing others doing it.

There are numerous operations that can produce the next hierarchical level. Examples include: abstraction and generalization operate on lower level objects to create higher level ones; compilation or aggregation can create higher level units; steps are combined together to create processes; going "meta" is in "thinking about thinking," moving from static to dynamic systems or linear dependency to mutual dependency also involve hierarchical transformations. Kegan notes that increasing complexity and sophistication moves (vertically) from entities to processes, from static to dynamic systems, and from dichotomous to dialectical relationships (Kegan 1994, p. 13).

Horizontal growth also follows a pattern in natural systems including human learning. The sequence is from single objects, to multiple independent objects, to multiple interacting objects, to massively interconnected object, and finally to an emergent whole that transitions to the next hierarchical level (see more in the Appendix). It makes intuitive sense that it is easy to learn a few more words (horizontal) but the leap to speaking sentences is comparatively momentous (which is not to say that it comes on line all of a sudden, i.e. children produce quasi-sentences first). And this difference is quantitative. If we wanted to measure language complexity we can count the size of vocabulary and the length of words, but no amount of increase in vocabulary will "equal" the shift from words to sentences.

Hierarchical Complexity (which is Commons' term, while other developmentalists use different terms) contributes to our analysis of authoring tool complexity in several ways. First, it ameliorates the "dimensionality issue" by providing another tool for organizing the plethora of complexity dimensions, i.e. according to horizontal and vertical differences in complexity, toward our goal of coordinating the complexities of tool vs. task vs. user; and in our goal to compare two (or more) tools (or tasks, or types of users). Second, because it is primarily a learning or developmental theory, it provides important insights into the effort and prerequisite knowledge a new user needs to use an authoring tool. Vertical growth is typically more difficult than horizontal growth, and the emergence of a new level of organization can come with some disequilibrium or dissonance, which in turn means there can be resistance or hesitancy.

We can begin with a rough characterization of the level of software tool complexity that a hypothetical user already has, and then ask whether the features and tasks of an authoring tool represent horizontal or vertical types of learning for the skill acquisition learning curve. We must not assume that new user skill level can be increased in any sort amount of time with something like a training intervention if vertical learning is involved.

Hierarchical Complexity and Epistemic Forms The analysis of tool/task/user complexity can proceed in two directions: more rigorous quantitative analysis, and more heuristic qualitative analysis. For our purposes we will focus on heuristic estimations. Our goal is to either start with a particular authoring tool/task and identify the communities of practice and training needs that will match the tool/task; *or*, starting with a target user group, to design the tool/task to match the estimated complexity of a community of practice. One can use the concepts introduced in this paper, including the dimensions of complexity, types of Epistemic Forms, and the distinction between horizontal and vertical differences in complexity, to make subjective shoot-from-the-hip assessments and inform design discussions as is usually done in software design. Alternatively, and left for others to carry forward, one can use these concepts to construct detailed quantitative metrics and formulas for calculating task/tool/knowledge complexity—but such in not necessary to make solid progress in matching tools/tasks to users.

Morrison and Collins mention the "epistemic complexity" of epistemic forms and games, but they do not define it precisely. What we contribute here is an attempt to link epistemic games to cognitive developmental theory in an attempt to create a grounded framework for assessing the relative complexity of epistemic forms/games, which will then provide a framework for describing the complexity of authoring tool features. These epistemic forms can be sequenced according to complexity level modeled on the levels mentioned in Hierarchical Complexity Theory (see Appendix), as shown in Table 3.

Figures 5 and 6 contain a series of figures illustrating these five complexity levels. Figure 5 summarizes the five levels of epistemic forms detailed in Fig. 6. In Fig. 5 we

Table 3 Epistemic forms organized by complexity level

Epistemic Form for Tool/Task/Mental Model	Complexity Level
• Text information fill-in boxes • Lists, choices, sliders, and check boxes	Simple Objects
• Forms, schemas, or templates • Tables and matrices • Hierarchies and trees	Abstractions & Mappings
• Scripts (with branches) • Equations and Boolean logic • Structural models: concept networks, boxology diagrams	Formal Systems
• Causal and constraint models (and using variables) • Behavioral/procedural models: If/then and rule-based procedural representations • (Authoring of) Decision trees, Bayesian Nets, etc.	Dynamic Systems
• Coordination of dynamic modules, e.g. complex interactions between expert, student, teaching modules, and dynamic use scenarios. • Design that takes into account emergent and chaotic interactions.	Architectures and Ecosystems (systems of dynamic systems)

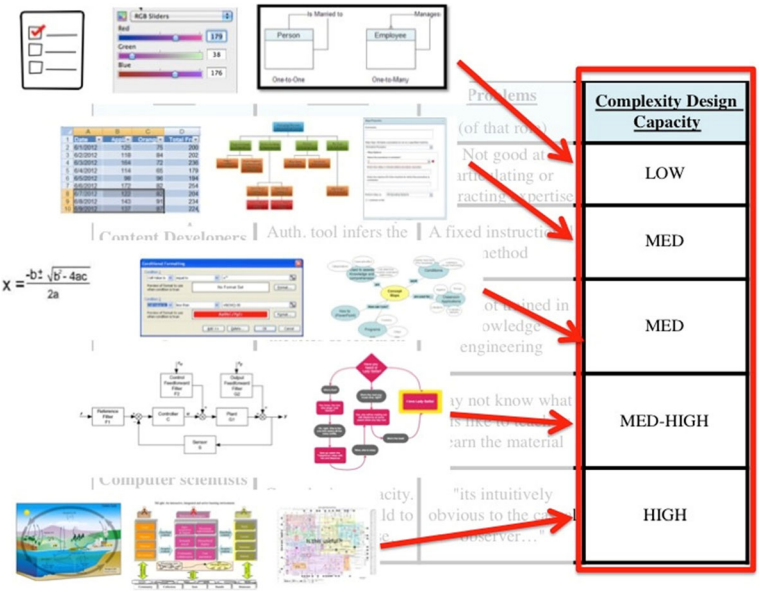

Fig. 5 Complexity levels of epistemic forms (overview)

link these complexity levels to the low/medium/high level of complexity associated with different categories of users from Table 2. Again, this mapping is a heuristic estimation that is intended to illustrate the type of analysis; no strong claims are made for the specific mappings.

Discussion

Beginning with a summary of my article on ITS authoring tool design, I described some of the challenges facing authoring tool designers and researchers today. Consonant with this Special Issue's theme of personal retrospectives on classic papers, I also included a narrative look at what brought me to authoring tools work, and mentioned that my academic journey since then has included interdisciplinary tributaries outside of ITS and educational technology per-se. The invitation to write this article has given me the happy opportunity to apply new frames of reference to an old topic. The reader hoping for definitive answers to questions about software complexity may have been disappointed—what I have done is exploratory theorizing to help frame important questions by suggesting certain theories, principles, and concepts amenable to ITS authoring tool R&D.

In this article I have explored some theoretical bases for assessing the appropriateness of ITS authoring tools, and *any* type of software artifact, to intended user communities. The analysis is based on general notions of complexity from Complexity Science and Hierarchical Complexity Theory. The importance of considering tools, tasks, user capacity, and community of practice in an integrated way was supported through the inclusion of the models of Activity Theory and Epistemic Forms.

Matching tool/task complexity to user/community complexity capacity is important because authoring tools are complex and expensive to build, and, using a "risk analysis"

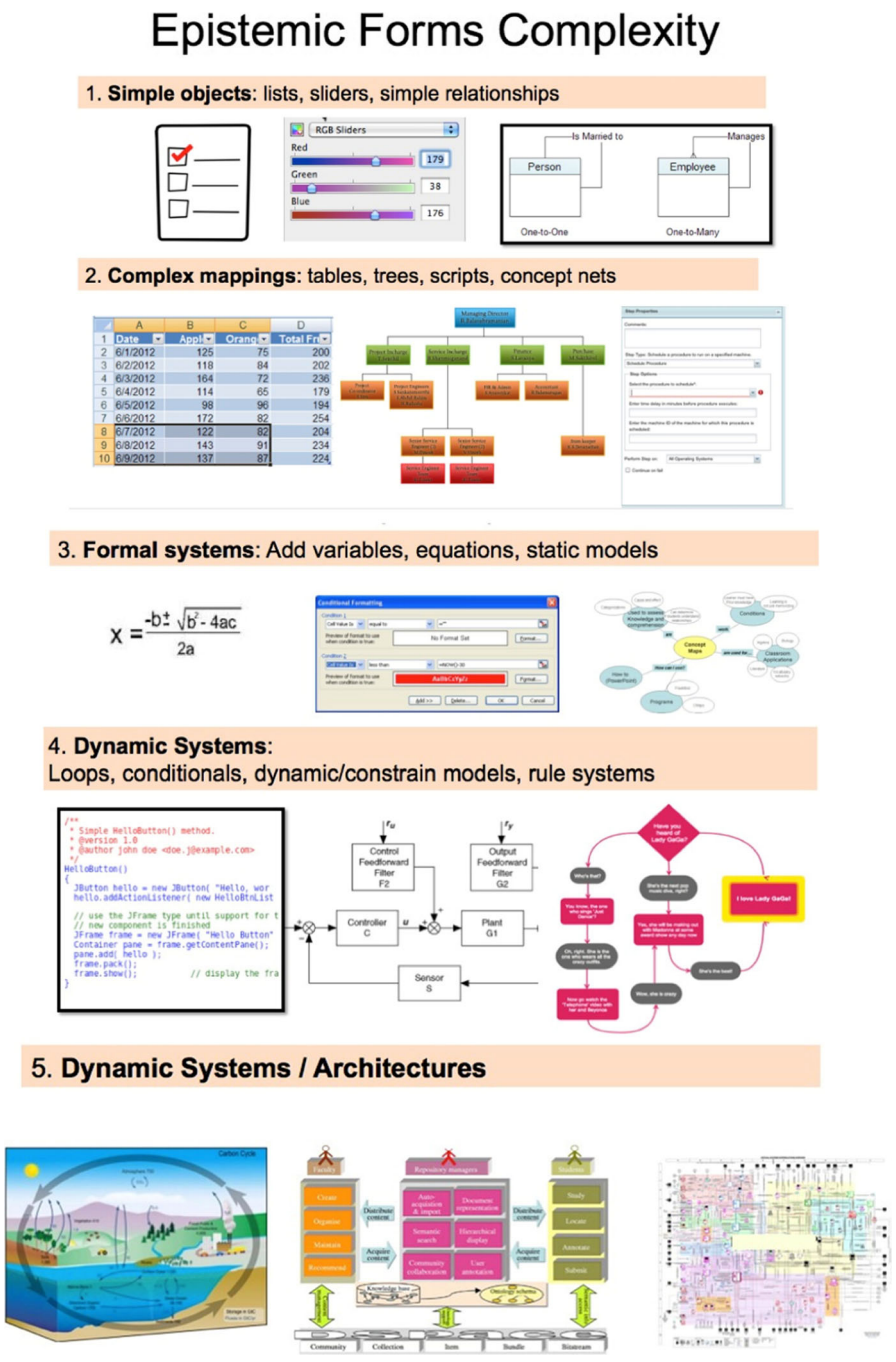

Fig. 6 Complexity levels of epistemic forms (details)

framework, we can say that the more expensive a system is to build, the larger the risk if user needs and capacities are not understood and anticipated. The design goal is to find the sweet spot where risk is acceptably low and expected value is relatively high. Oja's

(2010) study of improving usability in complex software systems concludes that systems should anticipate that projects usually involve a variety of roles and areas of expertise, and that interfaces should allow for the "distribution of tasks according to participant strengths" (p. 3800). Thus, the goal is not so much to match the affordances of an authoring tool to an intended user type, but to anticipate the *range* of user types involved in an ITS design and build tools that clearly meet the needs of each design role. Also, any plans for large scale adoption of authoring tools should include plans for learning and peer-mentoring within specific communication pathways in communities of learning.

The inclusion of Complexity Science and theories of dynamic systems in our narrative supports a bigger picture consideration of authoring that considers, not only how tools should be build to match user capacities, but the reciprocal evolution of tools and human capacities over longer periods of time. As Jerome Bruner notes "through using tools, man changes himself and his culture...human evolution is altered by man-made tools" (Bruner 1987). Thus tools can not only support the construction of advanced learning systems, but might also be designed to help users (especially instructors) more deeply understand and incorporate leading edge learning theories and mental models of the learning process (or build more adequate mental models of their content domain). We can move beyond seeing authoring tools primarily in terms of time and effort savings and consider their role in *empowering* content and pedagogy experts, including teachers; and in terms of propelling the evolution of computer-mediated learning in general.

Appendix: Neo-Piagetian Developmental Models: Skill Theory and Hierarchical Complexity Theory

In terms of this article as a personal retrospective, one question is "what have I learned since my seminal work on authoring tools that is applicable to the field?" Some of what I have learned is from my interdisciplinary dabbling in the areas of usability, Activity Theory, and Complexity Science, as discussed in this paper. But the most significant field that I have been exposed to in the last decade is adult developmental theories in the Neo_Piagetian tradition (from Fischer, Commons, Kegan, Cook-Greuter, and others, as discussed below). This literature could have significant impact in learning theory and educational technology R&D, and yet is hardly cited or known in these academic communities. These theories come from rigorous empirical studies and deeply inform questions about knowledge, learning and development, transfer, meta-cognition, reflective reasoning, socialization, and even motivation and ethics.[13] Thus I take pleasure in the opportunity to support a bit more familiarity to Neo_Piagetian theories within the ILS/CSCL/ITS communities. In this paper I focus on how these theories can inform the analysis of tool, task, and cognitive complexity, but in this Appendix I will say a bit more about these developmental theories, extending the description found in the section "Human Development and Hierarchical Complexity."

[13] Commons notes that "The Model of Hierarchical Complexity and Skill Theory...have ordered problem-solving tasks of various kinds, including": Social perspective-taking, Workplace organization, Political development, Political development, Writing, Epistemology, Algebra, Music, Animal intelligence, Counseling, and many more. See Commons and Richards 1984.

Neo-Piagetian developmental theories deepen and generalize early work by Piaget, Kohlberg, Perry and others. They are more nuanced and flexible than the original stage theories and, developmental scholars would say, adequately address problems earlier theories had with rigid stages, domain-specific learning, not accounting for individual differences, and the potential for cultural and other biases. They extend earlier work on childhood development into adult development in to post-formal-operational levels. They are also more compatible with contemporary dynamic systems, information science, and social-constructivist theories vs. earlier developmental theories. They also address the integration of bio/neuro/psycho/social/eco factors in human development more fully than prior theories (Knight and Sutton 2004; Demetrio et al. 2005; Dawson and Stein 2011).

As mentioned in the body of the paper, Neo-Piagetian developmental theories (henceforth "developmental theories") describe two general types of growth or learning, and thus two directions for increasing complexity, which can be called horizontal and vertical. Horizontal learning involves learning more of the same type of thing, for example, learning more plant species or violin concertos. Horizontal learning includes increasing differentiation among exemplars, for example, as one learns about more types of wine one becomes facile in the many ways that they differ, and gains nuance in noticing differences.

At some point, after many exemplars and much practice, learning takes a vertical leap to a new level of abstraction, integration, or consolidation. Just as biological cells and information networks tend to self-organize into larger wholes, ideas or knowledge itself appears to do something similar. This creates a qualitatively different form of understanding or skill with each vertical reintegration. In learning the component notes of a song, or the component actions of riding a bicycle, at some point the song or the action of riding emerges cognitively and is understood as a whole. The move from speaking individual words to sentences is a similar jump, as is the jump from learning addition and subtraction (horizontally related skills) to learning multiplication and division.

Cook-Greuter (2005) explains it this way: "most growth in adults is of the horizontal, expansion kind. People learn new skills, new methods, new facts, even new ways of organizing knowledge, but their current stage or mental model of the world remains the same…Developmental Theory, on the other hand, describes a sequence of how mental models themselves evolve over time. Each new level contains the previous ones as subsets. Each new level is both a new whole logic with its own coherence, and – at the same time – also a part of a larger, more complex meaning system." Robert Kegan describes development in terms of an evolving sequence of "distinctly different ways of making meaning" (1994, p. 90). Research has shown that such reorganizations or consolidations of horizontal knowledge are indeed non-linear events indicated by a sharp "spurts" (and then leveling off to another gradual increase) in skill level (Fischer and Yan 2002; Dawson-Tunik et al. 2005).

Neo-Piagetian (adult) developmental theories posit generalized developmental levels or tiers that grow over the lifespan but are also useful for the analysis of any specific human task, knowledge, or skill. The sequence of "complexity orders," which extends and refines the original work by Piaget, goes roughly: sensory-motor schema, conceptual categories and preoperational thinking, concrete operational thinking (representational); formal operational thinking (a. abstract objects, b. formal rules, c. systematic structures); and meta-principles and post-formal thinking (metasystematic principles and paradigmatic frames) (I have combined terms used in Skill Theory and Hierarchical Complexity Theory here). Development (in adults) happens in response to specific life demands, and skills in different domain or task areas evolve at different rates.

In vertical growth a new level of understanding is created through applying an operation upon, or "coordinating and transforming," the objects of the hierarchically lower layer. Each skill or knowledge at a given level consolidates a set of items at the prior level into a single whole, transcending and yet including them. In addition to the vertical growth mechanism of hierarchical inclusion, horizontal growth also follows a pattern. It starts with being able to recognize (or enact) a new type of object—you "see" or have a concept of something you have never known before. In seeing more of them one notices how they relate and begins to notice simple relationships between pairs, then to coordinate multiple objects (or actions). Interrelationships progress from individual and linear to interdependent and non-linear. At the final stage within a level objects are experienced in a dense network of elaborated interrelationships which takes on the wholeness of a new object at the next higher level. There is also a temporal element to the learning sequence within a level. We progress from engaging with a construct through afterthought and retrospective reflection, to increasingly being able to operate with it in real time. We also increasingly improve our ability to respond rapidly to dynamic and evolving contexts (until eventually our responses become automatic and unconscious). (These ideas are not unique to developmental theorists, and are reflected in other learning theories, such as Anderson's ACT model 1983).

Figure 7 shows the sequence of developmental levels in Fisher's skill theory (the most widely cited framework; and one that is compatible with Commons', Kegan's, and Dawson's frameworks). It shows a number of "tiers" representing the hierarchical level of the task or mental construct. Each builds upon the prior as mentioned above. Within each tier, or class of objects-to-be-operated upon, is the same sequence of operations: (1) single objects, (2) mappings (linear relationships) of objects, (3) systems (and non-leaner relationships) of objects; (4) systems of systems which organize (or chunk) into a single object at the next higher level.[14]

As an example, a child might first learn what a "lunch" is (a concrete tier object). They can recognize lunch and name it. As understanding progresses they coordinate lunch with other concrete constructs, perhaps soups and sandwiches, lunch times and locations such as school cafeterias, and are also learning what breakfast and dinner is (representational mappings). At the level of representational systems they understand and can talk about the system of breakfast-lunch-dinner, perhaps why certain foods or certain rules apply for each one, and can imagine related possibilities in the concrete realm ("what if we always ate at midnight—what would that be like?"). As a system is understood it is chunked into a schema at the next higher level, which is this case includes *abstractions* such as meal, etiquette, nutrition, etc. (e.g. concrete systems of systems become single abstractions). This framework is generic to all human tasks and skills (knowledge), and has been applied to algebra, reading, piano playing, tennis, parenting, leadership development, etc.

For Fischer and other developmentalists, mental growth is closely tied in to goal-directed activity in life. Skills are acquired only in response to "tasks" that one is challenged to succeed at as one interacts with artifacts and other individuals in authentic contexts. In this sense contemporary developmental theory is compatible with Activity Theory and socio-constructivist theories of learning (see section on Activity Theory).

[14] Within each tier are 4 numbered levels, but the highest level "systems" in any tier is equivalent to the most basic unit ("single") at the next tier—thus indicators such as "R4/A1" in the Figure.

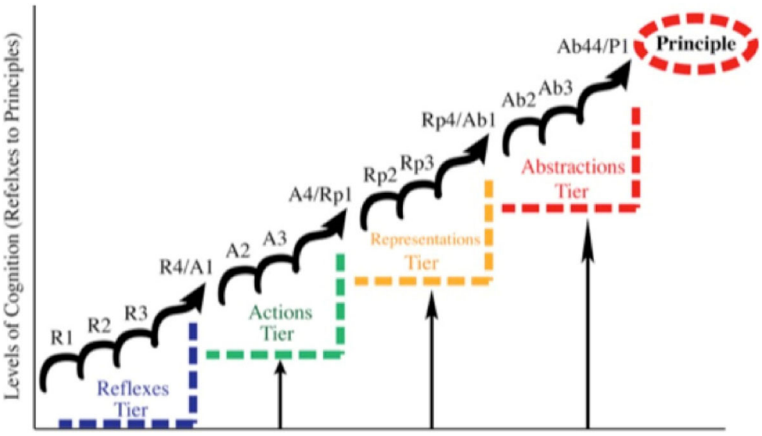

Fig. 7 Fisher's skill theory

Table 4 illustrates a series of children's verbalizations from different ages illustrating how the concept of "fun" develops through single (concrete) representations, representational mappings, representational systems, single abstractions, and abstract mappings. These illustrations are for verbal/cognitive domains. Commons and Pekker (2008) shows how these ideas can be applied generally to complex systems and artifacts. My goal in this paper is to apply these ideas to the design of interfaces and software-use tasks. What is important then is that the development of skills (and tasks) has a *semantic*/*conceptual* aspect, i.e. the types of objects in each successively more complex tier; and a *syntactic*/*structural* aspect, i.e. the four phases within each tier that describe how the elements are combined from simple to increasingly complex ways.

Developmental theory makes use of and parallels many aspects of Complexity Science. Developmental progressions in the horizontal direction include increase in these forms of complexity that have been mentioned: single to multiple items; independent to interacting items; static to dynamic contexts; linear to non-linear relationships; predicable/definitive to more unpredictable and fuzzy. Structural, dynamic, and perspectival modes of complexity increase, often in parallel.

Fischer's Skill Theory (1980) establishes a solid theoretical link between cognitive science and models from socio-technical fields such as Activity Theory.[15] Skill Theory frames the development (i.e. learning) of all human capacities, including intellectual, emotional, physical, musical, etc., in terms of "skills." Skills develop in response to, and only in response to, actual life "tasks" (i.e. situations calling for a response). In this framework, it does not make sense to describe a skill without describing the task, or general type of task, it is meant to address. In this model the analysis of cognitive capacities (learning, knowledge, etc.) is always coordinated with the analysis of tasks.[16]

[15] Fischer is one of the leading developmental theorists following the long lineage of neo-Piagetian theorists. His status within certain academic communities is on par with John Anderson's status in the Learning Sciences. Related and compatible frameworks include those from Commons (Commons and Richards 1984) and Dawson (Dawson and Stein 2011).

[16] Fischer is not a behaviorist. He does not deny the usefulness or realty of cognitive capacities that can not be directly measured. He merely grounds them in actual tasks. The tasks need not be physical and can be mental tasks, as they must be in moving from concrete to formal operational thinking (e.g. planning or mathematics).

Table 4 Development examples: the concept of fun

Single Ref. (unconnected list)	Fun is swinging on a swing. It's sliding on a slide.
Ref. Mapping (connections)	Fun is when Tommy and I put blocks together and then knock them down so that they make a loud noise that make us laugh.
Ref. System (interconnections)	Fun is different things. Sometimes I like to climb...that makes me...
Single Abstr. (unconnected list)	Fun is a way of enjoying yourself. It is a form of pleasure.
Abst. Mapping (connections)	There are a variety of ways that a person can have fun. Some people enjoy physical activities, like sports or just exercise. Some people...

This is compatible with Activity Theory and situated and Vygotsky-inspired learning theories that critique the separating cognition from activity in empirical or theoretical analysis.

References

Abelson, H., & Sussman, G. J. (1983). *Structure and interpretation of computer programs*. Cambridge: MIT Press.

Ainsworth, S., Major, N., Grimshaw, S., Hayes, M., Underwood, J., Williams, B. & Wood, D. (2003). REDEEM: simple intelligent tutoring systems from usable tools. Chapter 8 In MURRAY, T., BLESSING, S. & AINSWORTH, S. (Eds.). Authoring tools for advanced technology learning environments. Springer: Netherlands.

Aleven, V., & Sewall, J. (2010). Hands-on introduction to creating intelligent tutoring systems without programming using the cognitive tutor authoring tools (CTAT). In *Proceedings of the 9th International Conference of the Learning Sciences-Volume 2* (pp. 511–512). International Society of the Learning Sciences.

Aleven, V., McLaren, B. M., Sewall, J., & Koedinger, K. R. (2006). The cognitive tutor authoring tools (CTAT): Preliminary evaluation of efficiency gains. In I*ntelligent Tutoring Systems* (pp. 61–70). Springer Berlin Heidelberg.

Aleven, V., McLaren, B., Sewall, J., VAN Velsen, M., Popescu, O., Demi, S., Koedinger, K. (2015). An Effective Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors.

Anderson, J. (1983). *The architecture of cognition*. Cambridge: Harvard Univ. Press.

Benbya, H., & McKelvey, B. (2006). Toward a complexity theory of information systems development. *Information Technology & People*, *19*(1), 12–34.

Bereiter, C., & Scardamalia, M. (2006). Education for the knowledge age: design-centered models of teaching and instruction. In P. A. Alexander, & P. H. Winne (Eds.), *Handbook of educational psychology* (2nd ed., pp. 695–713). Mahwah: Lawrence Erlbaum Associates.

Brown, A. L., & Campione, J. C. (1996). Psychological theory and design of innovative learning environments: On procedures, principles, and systems. In L. Schauble, & R. Glaser (Eds.), *Innovations in learning: New environments for education* (pp. 289–325). Mahwah: Lawrence Erlbaum Associates.

Bruner, J. (1987/2004), 'Life as narrative', Social Research, 71: 691–710.

Byström, K., & Järvelin, K. (1995). Task complexity affects information seeking and use. *Information Processing and Management*, *31*(2), 191–213.

Campbell, D. J. (1988). Task complexity: a review and analysis. *Academy of Management Review*, *13*(1), 40–52.

Chilana, P. K., Wobbrock, J. O., & Ko, A. J. (2010). Understanding usability practices in complex domains. In Proceedings of the *SIGCHI Conference on Human Factors in Computing Systems* (pp. 2337–2346). ACM.

Cobb, P., Confrey, J., diSessa, A. Lehrer R. Schauble, L. (2003). Design experiments in educational research. Educational Researcher 2003; 32, 1.

Collins, A., & Ferguson, W. (1993). Epistemic forms and epistemic games: structures and strategies to guide inquiry. *Educational Psychologist*, *28*(1), 25–42.

Commons, M. L., & Pekker, A. (2008). Presenting the formal theory of hierarchical complexity. *World Futures: Journal of General Evolution*, *64*(5–7), 375–382.

Commons, M. L., & Pekker, A. (2009, unpublished). Hierarchical complexity and task difficulty. http://dareassociation.org/papers.php. Accessed Monday, November 30, 2009.

Commons, M. L., & Richards, F. A. (1984). A general model of stage theory. In M. L. Commons, F. A. Richards, & C. Armon (Eds.), *Beyond formal operations: Late adolescent and adult cognitive development* (pp. 120–141). New York: Praeger.

Commons, M. L., Trudeau, E. J., Stein, S. A., Richards, F. A., & Krause, S. R. (1998). Hierarchical complexity of tasks shows the existence of developmental stages. *Developmental Review*, *18*, 238–278.

Conklin, J. (2005). Wicked Problems & Social Complexity. Chapter 1 of *Dialogue Mapping*: *Building Shared Understanding of Wicked Problems*, Wiley.

Constantin, A., Pain, H., & Waller, A. (2013). Informing the design of an authoring tool for developing social stories. In *Human-Computer Interaction–INTERACT 2013* (pp. 546–553). Springer Berlin Heidelberg.

Cook-Greuter, S. R. (2000). Mature ego development: a gateway to ego transcendence. *Journal of Adult Development*, *7*(4), 227–240.

Cook-Greuter, S.R. (2005). Ego development: nine levels of increasing embrace. Available at www.cook-greuter.com.

Cristea, A. (2005). Authoring of adaptive hypermedia. *Journal of Educational Technology & Society*, *8*(3).

Dawson, T. L., & Stein, Z. (2011). We are all learning here: Cycles of research and application in adult development. Hoare, C. (Ed.). (2011). The Oxford handbook of reciprocal adult development and learning. Oxford: Oxford University Press.

Dawson-Tunik, T. L., Commons, M., Wilson, M., & Fischer, K. (2005). The shape of development. *The European Journal of Developmental Psychology, 2*, 163–196.

Demetriou, A., Efklides, A., & Shayer, M. (Eds.). (2005). *Neo-Piagetian theories of cognitive development*: *Implications and applications for education*. Routledge.

Engestrom, Y., Miettinen, R., & Punamaki, R.-L. (Eds.) (1999). *Perspectives on activity theory*. New York: Cambridge University Press.

Fischer, K. (1980). A theory of cognitive development: the control and construction of hierarchies of skills. *Psychological Review*, *87*(6), 477–531.

Fischer, K., & Yan, Z. (2002). The development of dynamic skill theory. In *Conceptions of development*: *Lessons from the laboratory*, 279-312.

Gentner, D., & Stevens, A. (Eds.) (1983). *Mental models*. Hillsdale: Lawrence Erlbaum Assoc.

Gersh, J. R., McKneely, J. A., & Remington, R. W. (2005). Cognitive engineering: understanding human interaction with complex systems. *Johns Hopkins APL Technical Digest*, *26*(4), 377–382.

Graesser, A. C., Chipman, P., Haynes, B. C., & Olney, A. (2005). AutoTutor: an intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions on Education*, *48*, 612–618.

Grünwald, P. D., & Vitányi, P. M. (2003). Kolmogorov complexity and information theory. With an interpretation in terms of questions and answers. *Journal of Logic, Language and Information*, *12*(4), 497–529.

Haynes, S. R., & Kannampallil, T. G. (2004). Learning, Performance, and Analysis Support for Complex Software Applications. *Proc. of the 3rd Ann. Workshop on HCI Research in MIS*, 30–34.

Heffernan, N., & Heffernan, C. (2014). The ASSISTments ecosystem: building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education*, *24*(4), 470–497.

Johnson, C. W. (2006). Why did that happen? Exploring the proliferation of barely usable software in healthcare systems. *Quality & Safety in Health Care*, *15*, i76–i81.

Johnson, W. L., & Valente, A. (2008). Tactical Language and Culture Training Systems: Using Artificial Intelligence to Teach Foreign Languages and Cultures. In AAAI (pp. 1632–1639).

Johnson-Laird, P. N. (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness*. Cambridge: Harvard University Press.

Jonassen, D., & Rohrer-Murphy, L. (1999). Activity theory as a framework for designing constructivist learning environments. *Educational Technology Research and Development*, *47*(1), 61–79.

Jordan, T., Andersson P., & Ringnér, H. (2013). The Spectrum of Responses to Complex Societal Issues: Reflections on Seven Years of Empirical Inquiry. *Integral Review*, February 2013, Vol. 9, No. 1.

Kegan, R. (1982). The evolving self. Cambridge: Harvard University Press.

Kegan, R. (1994). *In over our heads: The mental demands of modern life*. Cambridge: Harvard University Press.

Knight, C. C., & Sutton, R. E. (2004). Neo-Piagetian theory and research: enhancing pedagogical practice for educators of adults. *London Review of Education*, *2*(1), 47–60.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, *8*, 30–43.

Kumar, P., Samaddar, S. G., Samaddar, A. B., & Misra, A. K. (2010, June). Extending IEEE LTSA e-Learning framework in secured SOA environment. In *Education Technology and Computer* (*ICETC*), *2010 2nd International Conference* (Vol. 2, pp. V2-136). IEEE.

Mirel, B. (2004). *Interaction design for complex problem solving*. San Francisco: Morgan Kaufman.

Mitrovic, A. (2012). Fifteen years of constraint-based tutors: what we have achieved and where we are going. *User Modeling and User-Adapted Interaction*, *22*(1–2), 39–72.

Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., & McGuigan, N. (2009). ASPIRE: an authoring system and deployment environment for constraint-based tutors. *Artificial Intelligence in Education*, *19*(2), 155–188.

Morrison, D., & Collins, A. (1995). Epistemic fluency and constructivist learning environments. *Educational Technology*, *35*(5), 39–45.

Murray, T. (1996). Having It All, Maybe: Design Tradeoffs in ITS Authoring Tools. In *Intelligent Tutoring Systems*: *Third International Conference*, *ITS'96*, Montreal, Canada, June 12–14, 1996. Proceedings (Vol. 1086, p. 93). Springer.

Murray, T. (1999). Authoring intelligent tutoring systems: analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, *10*(1), 98–129.

Murray, T. (2003a). An overview of intelligent tutoring system authoring tools: updated analysis of the state of the art. In T. Murray, S. Blessing, S. Ainsworth (Eds.), *Authoring tools for advanced technology learning environments*. Netherlands: Springer.

Murray, T. (2003b). Eon: Authoring tools for content, instructional strategy, student model, and interface design. In T. Murray, S. Blessing, S. Ainsworth (Eds.), *Authoring tools for advanced technology learning environments*. Netherlands: Springer.

Murray, T. (2004). Design tradeoffs in usability and power for advanced educational software authoring tools. *Educational Technology Journal*, *2004*, 10–16.

Murray, T., & Woolf, B. (1992). Tools for teacher participation in ITS design. In Frasson, Gauthier, & McCalla (Eds.), *Intelligent Tutoring Systems, Second Int. Conf.* (pp. 593–600). New York: Springer Verlag.

Murray, T., Blessing, S., & Ainsworth, S. (Eds.) (2003). *Authoring tools for advanced technology learning environments: Toward cost-effective adaptive, interactive, and intelligent educational software*. Netherlands: Springer.

Nielsen, J. (1993). *Usability engineering*. AP Professional: Boston.

Nielsen, J. (1994). Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 152–158). ACM.

Norman, D. (1988). *The design of everyday things*. NY: Doubleday.

Oja, M. K. (2010). Designing for collaboration: improving usability of complex software systems. In *CHI'10 extended abstracts on human factors in computing systems* (pp. 3799–3804). Chicago: ACM.

Olsen, J. K., Belenky, D. M., Aleven, V., & Rummel, N. (2013). Intelligent Tutoring Systems for Collaborative Learning: Enhancements to Authoring Tools. In *Artificial Intelligence in Education* (pp. 900–903). Springer Berlin Heidelberg.

Piaget, J. (1972). *The principles of genetic epistemology*. NY: Basic Books.

Razzaq, L., Patvarczki, J., Almeida, S. F., Vartak, M., Feng, M., Heffernan, N. T., & Koedinger, K. R. (2009). The assistment builder: *supporting the life cycle of tutoring system content creation*. *IEEE Transactions on Learning Technologies*, *2*(2), 157–166.

Ritter, S. (2015). Authoring for the product lifecycle. Available from the author at Carnegie Learning.

Ritter, S., & Blessing, S. (1998). Authoring tools for component-based learning environments. *The Journal of the Learning Sciences*, *7*(1), 107–132.

Shaffer, D. W. (2006). Epistemic frames for epistemic games. *Computers & Education*, *46*(3), 223–234.

Sitaram, S., & Mostow, J. (2012). Mining data from project LISTEN's reading tutor to analyze development of children's oral reading prosody. In Proceedings of the 25th Florida artificial intelligence research society conference (FLAIRS-25), 478–483. Marco Island, Florida.

Soloway, E., Guzdial, M., & Hay, K. E. (1994). Learner- centered design: the challenge for HCI in the 21st century. *Interactions*, *1*(2), 36–48.

Sottilare, R. A., Brawner, K. W., Goldberg, B. S., & Holden, H. K. (2012). The generalized intelligent framework for tutoring (GIFT). *Orlando*, *FL*: *US Army Research Laboratory–Human Research & Engineering Directorate* (*ARL-HRED*).

Sottilare, R., Graesser, A., Hu, X. & Goldberg, B. (2014). *Design recommendations for intelligent tutoring systems*: *volume 2*: *instructional management*. U.S. Army Research Laboratory Human Research & Engineering Directorate.

Specht, M. (2012). E-Learning Authoring Tools. In *Encyclopedia of the Sciences of Learning* (pp. 1111–1113). Springer US.

Stahl, G. (2006). *Group cognition: Computer support for building collaborative knowledge*. Cambridge: MIT Press.

Suraweera, P., Mitrovic, A., & Martin, B. (2010). Widening the knowledge acquisition bottleneck for constraint-based tutors. *International Journal of Artificial Intelligence in Education*, *20*(2), 137–173.

Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., … Wintersgill, M. (2005). The Andes physics tutoring system: lessons learned. *International Journal of Artificial Intelligence in Education*, *15*(3), 147–204.

Woolf, B. P. (2010). *Building intelligent interactive tutors*: *Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann.

Woolf, B. & McDonald, D. (1984). Design issues in building a computer tutor. *IEEE Computer*, Sept. 1984.

Zhang, J., Scardamalia, M., Reeve, R., & Messina, R. (2009). Designs for collective cognitive responsibility in knowledge-building communities. *The Journal of the Learning Sciences*, *18*(1), 7–44.