Human-in-the-Loop Challenges for Entity Matching: A Midterm Report

AnHai Doan, Adel Ardalan, Jeffrey Ballard, Sanjib Das, Yash Govind, Pradap Konda
Han Li, Sidharth Mudgal, Erik Paulson, Paul Suganthan G.C., Haojun Zhang
University of Wisconsin-Madison

ABSTRACT

Entity matching (EM) has been a long-standing challenge in data management. In the past few years we have started two major projects on EM (Magellan and Corleone/Falcon). These projects have raised many human-in-the-loop (HIL) challenges. In this paper we discuss these challenges. In particular, we show how these challenges forced us to revise our solution architecture, from a typical RDBMS-style architecture to a very human-centric one, in which human users are first-class objects driving the EM process, using tools at pain-point places. We discuss how such solution architectures can be viewed as combining "tools in the loop" with "human in the loop". Finally, we discuss lessons learned which can potentially be applied to other problem settings. We also hope that more researchers will investigate EM, as it can be a rich "playground" for HIL research.

1. INTRODUCTION

Entity matching (EM) finds data instances that refer to the same real-world entity, such as tuples a_1 and b_1 in Figure 1. This problem has been an important challenge in data management [4, 6], and will become even more so in the age of Big Data and data science.

In the past few years, we have started two major projects on EM. The first project, Magellan, seeks to build an EM management system that helps users with all aspects of the EM process [14]. The second project, Corleone/Falcon, seeks to build a cloud-based EM service that crowdsources the entire EM process, requiring no developer in the loop [8, 5]. These projects have produced software that has been used extensively in data science classes and at several organizations, e.g., Marshfield Clinic [16], Johnson Controls, Recruit Holdings, WalmartLabs, a non-profit organization, and in several projects in social sciences at UW-Madison.

These projects have raised many human-in-the-loop (HIL) challenges (in addition to many algorithmic and system ones). Many of these challenges are interesting, difficult, and have significantly changed our perceptions of what HIL means and how it should be addressed.

HILDA'17, May 14, 2017, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3077257.3077268

Table A

	Name	City	State
a_1	Dave Smith	Madison	WI
a_2	Joe Wilson	San Jose	CA
a_3	Dan Smith	Middleton	WI

Table B

	Name	City	State
b_1	David D. Smith	Madison	WI
b_2	Daniel W. Smith	Middleton	WI

Matches: (a_1, b_1), (a_3, b_2)

Figure 1: An example of matching two tables.

The papers [14, 8, 5] describe the above projects in detail, but focus on the motivations and the algorithmic/system challenges. In this paper, we discuss the HIL challenges. We begin with the Magellan project. We show how HIL challenges forced us to keep revising our solution architecture, from a typical RDBMS-style one (Figure 2) to a very human-centric one (Figure 6), in which human users are first-class objects driving the EM process, using tools at pain-point places. We discuss how such solution architectures can be viewed as combining "tools in the loop" with "human in the loop". We also discuss other surprising findings. For example, certain tasks that we thought trivial, such as labeling a data set and sampling, turn out to be quite difficult. As another example, it turns out that in many cases, at the start of the EM process, the user knows little about the problem (e.g., what it means to be a match), the data, and the tools' capabilities. This significantly complicates the solution architecture and changes our thinking about how to "shepherd" the user through the EM process.

Next, we discuss HIL challenges in the Corleone/Falcon project. These include the need to solve collaborative EM, to enable effective communications among the participants (to share knowledge about the problem/data/tools), and to develop more expressive UIs.

Finally, we discuss lessons learned regarding HIL challenges, which can potentially be applied to other problem settings. We also hope that the challenges described here will encourage more researchers to investigate EM, as it can be a rich "playground" for HIL research.

2. OUR EXPERIENCE

We now discuss HIL challenges raised by our recent EM projects. We focus on Magellan [14], which has raised most of such challenges. But we will also briefly discuss HIL challenges in the Corleone/Falcon project.

2.1 HIL Challenges in the Magellan Project

We started Magellan in 2015 to study data integration (DI). In contrast to most existing DI works, which develop DI algorithms, we decided to study how to develop *DI systems*, as we believe such systems are necessary for signifi-

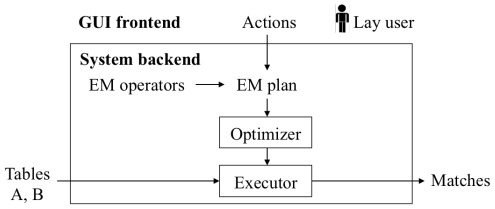

Figure 2: The RDBMS-style system architecture initially adopted by the Magellan project.

cantly advancing the field. Several efforts to develop DI systems already exist (e.g., [22]), but focus on supporting DI pipelines of *multiple* tasks (e.g., schema matching, schema integration, EM, etc.). We decided to focus on just one task, EM, so that we can study it as deeply as possible. Within EM, we further limited our attention to the scenario of matching two tables A and B, as illustrated in Figure 1.

Building an Initial EM System: At the start, we worked with WalmartLabs to build an EM system for their analysts. The analysts wanted to use the system to quickly create and execute EM workflows. They prefer to do so using a GUI, because many of them are lay users with little CS knowledge. We started out building this EM system in an RDBMS fashion, in a way that is very similar to many existing prototype EM systems (e.g., see those listed in [14]).

Figure 2 shows the system architecture. To explain, note that when matching two tables, users often execute a blocking step followed by a matching step. In *the blocking step*, the user employs heuristics to quickly remove obviously non-matched tuple pairs (e.g., persons residing in different states, see Figure 1). In *the matching step*, the user applies a matcher to the remaining pairs to predict matches [14].

As a result, we consider EM workflows consisting of the above two steps, and define actions that the user can perform on the GUI to create such a workflow (e.g., selecting a built-in blocker, writing a rule-based blocker, selecting a built-in matcher, etc.). In the system backend, such actions are translated into an EM plan composed of operators (e.g., to execute a rule-based blocker, such a plan executes an operator that converts tuple pairs into feature vectors, then another operator that applies the blocking rules to these feature vectors, etc.). The EM plan is then optimized and executed, to produce the matches (see Figure 2).

Limitations: We soon realized two major problems with this system architecture. First, users rarely create an EM workflow in "one shot". Instead, they often want to explore, debug, try "what-if" scenarios, then modify the workflow, in an iterative fashion. Just figuring out how to support these actions at the "operational" level (e.g., what does it even mean to support a power user in debugging?) is already quite difficult. Trying to figure out how to support them at the GUI level for lay users adds another level of complexity.

The second problem is that our current EM system architecture (Figure 2) is not adequate. In this architecture, we assume user actions on the GUI will be translated into an underlying EM plan, which is optimized and executed. But many actions, such as exploration, debugging, trying "what-if" scenarios, do not fit this assumption, i.e., they can not be modeled as a part of an EM plan. Exploration for instance is to help the user *understand* the data better; it cannot be modeled as an operator in an EM plan.

To address these, we made the following two decisions,

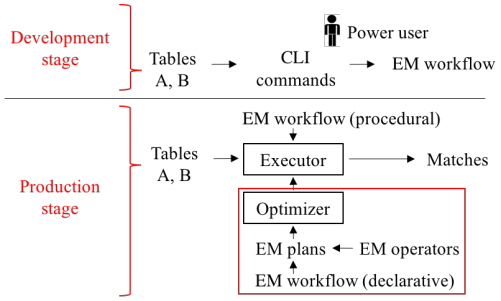

Figure 3: The revised system architecture consisting of a development stage and a production stage.

which led to the revised system architecture in Figure 3.

(a) Distinguish the Development & Production Stages: We assume the user will work in two stages: development and production (Figure 3). In *the development stage* the user develops a good EM workflow, i.e., one with high matching accuracy. This is typically done using data samples. The user can perform whichever actions he/she deems useful (e.g., exploration, debugging, cleaning, etc.), in order to obtain a good EM workflow. These actions are shown as CLI (i.e., command-line interface) commands in Figure 3.

In *the production stage* the user applies the EM workflow to the entirety of data, i.e., Tables A and B, to find matches. Since this data is often large, a major concern is scaling. Other concerns include quality monitoring, logging, crash recovery, etc. The default is that the EM workflow will be encoded procedurally. Alternatively, as shown in the red box in Figure 3, the user may specify this workflow declaratively. The system translates it into an EM plan composed of operators, then optimizes and executes the plan.

While this looks obvious in retrospect, we note that the vast majority of work on DI systems do not distinguish these two stages, even though they are very different by nature and require different solutions. (In fact, sometimes it is not even clear which stage a particular work is targeting.) Further, most works have addressed only the production stage. Very few works (e.g., Potter Wheel, Trifacta [20, 11]) address the development stage. Addressing this stage however is critical, because the user must start in this stage to come up with an EM workflow, *before* he/she can execute it. As a result, in this paper, we will focus on the development stage (but note that there are also many HIL challenges in the production stage).

(b) Start with Power Users in Command-Line Settings: WalmartLabs wanted an EM system for lay users. But as discussed earlier, we do not yet even know how to build such a system for *power users:* those who may not be DI experts but know how to program (e.g., data scientists). For example, we do not yet know how best to support exploration, debugging, etc.

As a result, we decided to start by building a system for *power users*, in a command-line (CLI) setting. This frees us from the distractions of having to deal with various limitations (e.g., the limited set of actions a *lay user* can do, the limitation of a GUI), to allow us to focus completely on figuring out how to support actions such as debugging at the "operational" level. As we gain a better understanding of what it means to help power users, we will build on that to consider more complex settings, such as lay users on a

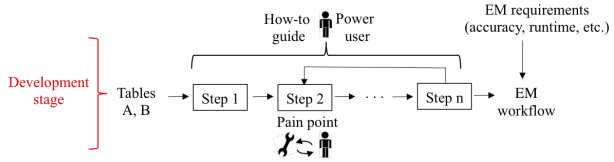

Figure 4: A revised version of the development stage of the EM system architecture in Figure 3.

GUI, collaboration, crowdsourcing, etc. This is analogous to developing assembly languages, then building on them to develop higher-level CS languages.

Using the revised system architecture in Figure 3, the Magellan project was able to move along at a faster pace. But it raised many new interesting HIL challenges (which in turn helped us refine the above architecture).

Need a Step-by-Step & End-to-End How-To Guide: One of the early HIL challenges we ran into is how to best help a user reach a matching goal. For example, what if a user wants to match two tables with 90% precision and 80% recall? Such EM scenarios are very common, and yet, surprisingly, there is no guidance today on what a user should do. How should he/she start? Should the user consider a learning-based EM approach first? Or rule-based? Or a combination of both? What happens if after extensive efforts with learning/rule-based approaches, the user still cannot reach the desired accuracy?

To address this problem, we propose to develop a detailed how-to guide, which tells the user how to perform EM, step by step, end to end (e.g., telling him/her how to go from the two input tables to matches satisfying the required accuracy). Note that such a guide does not assume the existence of any tool yet. It should be such that if the user is a power user and is willing to program, the user can easily follow the guide to perform EM end to end.

Develop Tools for "Pain Points" in the Guide: Given a how-to guide, we should carefully examine it to find "pain points", i.e., steps that are mundane, repetitive, or consume a lot of user time. Examples include sampling smaller tables from Tables A and B, profiling the tables, debugging a blocker, etc. We should develop (algorithms captured as) tools for these pain points. Some tools can be completely automatic (e.g., sampling), while others (e.g., debugging) are likely to engage humans.

Combine Tools in the Loop with Human in the Loop: The above decisions led to a revised architecture for the development stage as shown in Figure 4 (recall that we only focus on the development stage for now). In this revision, a how-to guide tells a power user U how to create an EM workflow that satisfies the EM requirements (e.g., regarding accuracy, runtime, etc.). The guide specifies a step-by-step procedure for the user to execute (e.g., Steps 1-n in Figure 4). The user executes "pain point" steps (e.g., Step 2) using tools, and the rest of the steps (e.g., Step 1, Step n, etc.) by hand. Thus, these tools can be viewed as "tools in a loop driven by a human user". Some tools engage user U, in an iterative fashion (e.g., Step 2). Thus, user U can be viewed as "a human in a loop driven by a tool".

As described, the above revised EM architecture combines "tools in the loop" with "human in the loop". Both kinds of loop are necessary. The outer loop (driven by the user

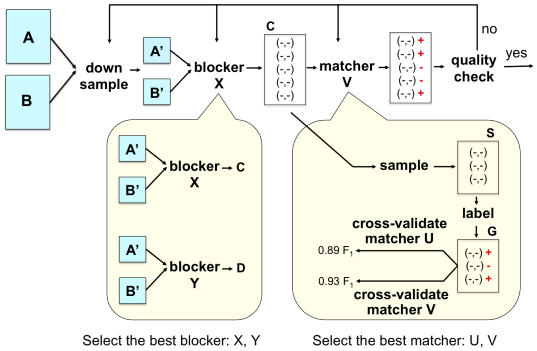

Figure 5: The main steps of a sample how-to guide for EM using supervised learning.

using a how-to guide) is needed because we cannot (yet) fully automate the end-to-end EM process. The inner loops (where tools engage the user) are needed because tools often need input from the user to maximize its accuracy, among others.

Develop a How-To Guide for a Concrete Scenario: After revising the development stage, as described above, we tried to "instantiate" it for the common EM scenario where the user wants to use supervised learning to match. We began by sketching out the main steps of the how-to guide. To do so, we observed how real users perform EM, observed how students in our classes did EM (e.g., what issues they struggled with), and did EM ourselves.

Figure 5 shows these steps. Suppose user U wants to match two tables A and B, each having 1 million tuples. Trying to find an accurate workflow using these two tables would be too time consuming, because they are too big. Hence, U will first "down sample" the two tables to obtain two smaller tables A' and B', each having 100K tuples, say (see the figure).

Next, suppose the EM system provides two blockers X and Y. Then U experiments with these blockers (e.g., executing both on Tables A' and B' and examining their output) to select the blocker judged the best (according to some criterion). Suppose U selects blocker X. Then next, U executes X on Tables A' and B' to obtain a set of candidate tuple pairs C.

Next, U takes a sample S from C, and labels the pairs in S as "match"/"no-match" (see the figure). Let the labeled set be G, and suppose the EM system provides two learning-based matchers U and V (e.g., decision trees, logistic regression). Then U uses the labeled set G to perform cross validation for U and V. Suppose V produces higher matching accuracy (such as F_1 score of 0.93, see the figure). Then U selects V as the matcher, and applies V to the set C to predict "match"/"no-match", shown as "+" or "-" in the figure. Finally, U may perform quality check (by examining a sample of the predictions), then go back and debug and modify the previous steps as appropriate. This continues until U is satisfied with the accuracy of the EM workflow.

The above how-to guide is relatively straightforward, but does not provide enough details for the user. So next we tried to refine this guide and develop tools for the pain points. Surprisingly, these have turned out to be quite difficult, as we elaborate below.

Need Tools for Debugging and Explaining: We will first discuss the challenges in developing tools, then build

on those to discuss refining the how-to guide. As expected, when a user follows the above how-to guide, a major pain point is debugging, e.g., debugging blockers and matchers, debugging the labels, etc. These steps are difficult and tedious, and users badly need debugging tools. We have developed a tool to debug blockers and a tool to debug the labeled tuple pairs, and are developing tools to debug matchers. For example, suppose that the user has created a blocker Q, and that when applied to Tables A' and B', Q produces the set of tuple pairs C. Then $D = A' \times B' \setminus C$ is the set of all tuple pairs "killed off" by Q. Our debugging tool finds true matches in D, so that the user can inspect these matches to understand whether Q kills off too many matches and how to modify Q to prevent that. To find matches in D, the tool uses multiple inexpensive "detectors" to quickly find candidate matches in D, then uses a more expensive "evaluator" (which performs active learning with the user in the loop) to evaluate this set of candidates, to find true matches. The tool then explains to the user why these matches are killed off by blocker Q, so that the user can improve Q.

As another example, we have recently developed a deep learning-based matcher for EM. While highly accurate in certain cases, this matcher has turned out to be very difficult to debug and explain, as expected. Yet, we cannot use it effectively until we have such tools (e.g., so that we can tune it to trade off between precision and recall).

Overall, we have found that (a) developing tools for debugging and explaining is critical, (b) virtually all such tools need "human in the loop", and (c) how to engage such humans effectively is still a difficult challenge.

Even Simple Steps Turned Out to Be Difficult: We found that several steps (in the how-to guide) thought trivial turned out to be anything but. Consider for example the step of labeling the tuple pairs in set S (see Figure 5). Suppose these tuples describe restaurants. A user U may start out labeling only tuple pairs describing restaurants *at the same location* as "match", e.g., (Saigon Noodle, 321 Main St, Madison, WI) and (SG Noodle, 321 Main, Madison, WI).

Suppose while labeling the first few hundred pairs, user U saw many restaurants that are *at different locations, but of the same chain*, e.g., (Chipotle, 426 Farewell, Madison, WI) and (Chipotle, 34 Main St, Madison, WI). U discussed this with the business owner, who decided that such restaurants should also match. The problem however is that up until then U had labeled all such restaurants as "no-match". So now U would need to re-examine all tuple pairs that had been labeled, to find and fix such cases, a tedious process. This problem becomes much worse if two users label the set S, each labeling one half, say. Both users start with the same match definition, but may revise them differently during the labeling process, and the inconsistency is detected only at the end, when they try to merge the two labeled halves.

Fundamentally, this problem arises because trying to define *a priori* what should be a match turns out to be quite difficult in many cases. Often users start with an initial match definition, then revise it along the way, as they understand the problem and the data better (see more below).

As another example, the step of taking a sample S (so that the user can label it) also turned out to be difficult. Recall that C is the set of all tuple pairs surviving the blocker (see Figure 5). If C is large, the fraction of C that is true matches may be small. As a result, if we take S to be a *random* sample from C, S may contain very few matches, rendering learning on S ineffective. To address this problem, we are currently exploring a solution that requires user U to repeatedly sample, label, then perform additional blocking if necessary, along the line of the solution in [8].

Understanding Problem/Data/Tools Is a Process: We found that very often at the start, the user has very little knowledge about the problem, the data, and the capabilities of the tools, and that he/she gradually gains more knowledge about these during the EM process, by going through multiple iterations. This finding is somewhat surprising. For example, the vast majority of EM work implicitly assume that the user knows the match definition. Our experience suggests that this is often *not* the case (as we discussed earlier). That is, the user often thinks that he/she knows the match definition. But as the user works more with the data, he/she uncovers cases that force him/her to rethink and revise the match definition.

As another example, the user often starts with very little knowledge about the data in Tables A and B. Then during the EM process he/she gradually learns more about whether the data is dirty, how pervasive are missing values, are there any regions of the data that are simply incorrect, etc. This in turn helps him/her revise the EM workflow. As yet another example, the user often starts with no idea on whether a particular tool will work well on a particular dataset. Only after experimenting in multiple iterations that he/she gain a much better understanding, which may lead the user to make decisions such as replacing the tool, or augmenting it with some other tools, etc.

Must Output Knowledge about Problem/Data/Tools: As discussed earlier, at the start of the development stage, the user often knows very little about the problem definition, the data profile, and the tools' capabilities. He/she often gains far more knowledge about these along the way. We found that at the end of the development process, it is often highly desirable for the user to output not just a good EM workflow, but also knowledge about the problem/data/tools.

For example, besides a good EM workflow, the user U can produce a report listing possible match definitions, the selected definition, and the reasons for selecting it. U can produce another report listing various problems with the data, how they can affect EM, actions that should be avoided, as well as actions that can be taken to fix the problems. For example, this document can say that there are many missing and dirty values in column "state", hence do not do blocking using "state" (e.g., drop all tuple pairs whose tuples disagree on "state"). Finally, the user can produce a report discussing the capabilities of the tools on the current tables and actions taken based on those (e.g., learning-based matchers do not appear to work well here for such and such reasons, and hence rule-based matching is added, to reach the desired EM accuracy).

Having such reports is tremendously useful because, as mentioned multiple times, EM is often an iterative rather than "one-shot" process. Even after an EM workflow has been pushed into production, problems often occur, which necessitates working in the development stage again to repair/fine tune the EM workflow. Having access to such reports makes this process much easier (especially in the case the user has moved on and a new data scientist is now debugging the EM process).

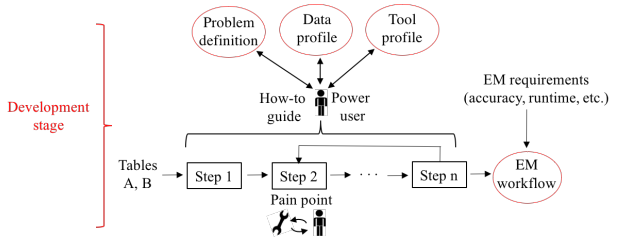

Figure 6: A revised version of the development stage of the EM system architecture in Figure 4.

A Revised Development Stage: The above findings led to the revised development stage in Figure 6. Here, the goal is not just to output a good EM workflow, but also to gain as much knowledge as possible about the problem definition, data profile, and tool profile, and output those as well (the red circles in the figure indicate the outputs). Of course, the knowledge about problem/data/tools is also used along the way to help find a good EM workflow.

We are currently working on realizing this architecture. It turns out to be quite difficult, because we have to revise the how-to guide and it is still unclear how to do that. For example, presumably at the start of the development stage, the user should be able to explore alternative matching definitions, and select one, as quickly as possible, because having the precise matching definition helps subsequent steps. However, it still remains unclear how best to help the user explore alternative matching definitions. This raises both HIL and algorithmic challenges.

2.2 HIL Challenges in Corleone/Falcon

We now discuss HIL challenges in the Corleone/Falcon project. For space reasons we can only briefly discuss these challenges, deferring more details to a later paper.

Our three most important HIL findings in this project are as follows. First, we found that in many EM settings there is actually a team of people wanting to work on the problem. So how to collaboratively solve an EM problem (e.g., collaboratively label a data set, debug, clean the data, etc.) is important, and yet this has received very little attention in the EM literature. Note that collaborative EM is different from crowdsourced EM. In the former the team may be small but each team member is assumed to be quite capable of many (complex) actions. In the latter, each crowd worker is assumed to do only a few restricted actions, e.g., labeling a tuple pair.

Second, if multiple persons work on an EM problem, we do not yet know how to coordinate so that they can efficiently communicate their findings about the problem/data/tools, and can converge at the end (e.g., to a single match definition). This problem arises in the crowdsourced EM context as well. A user may start with an initial match definition. Crowd workers may be confused and want clarification, etc. Currently there is also no good way to facilitate such communications between the user and crowd workers.

Finally, the feedback we have received from many users is that the current UIs are too limited. More expressive UIs are highly desirable. For example, when we perform crowdsourced EM with in-house experts, they are still limited to just labeling tuple pairs (as the vast majority of current crowdsourced EM works also do). They find this frustrating. For example, if they see dirty tuples, they may want to clean those on the flight, but today they cannot. As another example, some indicated that an UI that shows them a cluster of tuples (that are supposed to match) may help them "label" data faster than showing one tuple pair at a time. Thus, an interesting HIL challenge is to explore better UIs for in-house experts. Another challenge is to examine whether the current UI (of labeling tuple pairs) is also too limited for crowd workers. Perhaps more expressive UIs would make crowd workers more efficient as well.

3. LESSONS LEARNED

The main lessons we have learned so far are as follows. First, we believe it is important to focus on a core problem (e.g., EM, schema matching, information extraction, managing missing values), so that we can examine it in depth. As we gain more knowledge about solving each individual problem, we can build on that to solve these problems jointly.

Second, it may be a good idea to focus on power users. Again, as we gain more knowledge about how to solve the problem for power users, we can build on that to solve the problem for lay users, collaborative settings, crowdsourcing settings, etc. As our current EM work suggests, we still have very limited understanding of how to effectively solve a single core problem (such as EM), even for power users.

Third, we may want to broaden our focus beyond developing algorithms, to developing systems as well. We believe the system "angle" will help us "unearth" problems, evaluate algorithms, make practical impacts, and significantly advance the field. We certainly have learned a tremendous amount from our attempts to build EM systems and use them in our classes and at companies. These attempts have completely changed our perspectives and thinking about EM and HIL.

Fourth, we cannot build such systems by "stringing together" a set of algorithms (e.g., several blockers, several matchers). We really need to change the perspective and think from a human user point of view. That is, we need to design the system such that it supports *all the things* that a human user may need to do, to solve the problem. To do so, we need to examine the *end-to-end* process of how a human user may solve the problem, distinguish the development and production stages, and design a how-to guide for the user. As our work has shown, designing effective how-to guides poses serious challenges.

Fifth, given the how-to guide, we can then identify pain points and build tools to address those. This may really change what we are doing today. For example, designing yet another EM algorithm that improves EM accuracy by a few percentage F_1 may not address the true pain points, but developing tools to help the labeling process may.

There is some evidence to suggest that this "how-to guide and tools for pain points" approach is effective. Open-source DI tools such as those in the Python/R ecosystems have been widely used. This is due partly to the fact that many of such tools are developed to address some pain points encountered by the tools' creator; a future user is likely to encounter the same pain points, and thus is likely to find such tools useful.

Sixth, developing tools for pain points likely requires engaging the human user, in an iterative fashion. This gives rise to a solution architecture that combines "tools in the loop" (of executing the how-to guide) with "human in the loop" (of the tools), as shown in Figure 6. We suspect that this architecture can be useful for many other problem settings, not just for EM.

Seventh, there is strong evidence (in our work and elsewhere) that at the start, the user often knows very little about the problem definition/data/tool capabilities, that the user tries to gain more knowledge about these along the way, and that it is highly desirable to output this knowledge too (e.g., in form of reports). This again complicates the solution architecture (e.g., see Figure 6), but we believe makes it far more practical. How to help the user gain such knowledge effectively is still very much an open question, and is a major HIL challenge.

It is worth noting that a solution architecture such as shown in Figure 6 is very heavily human-centric, with humans being first-class objects driving the entire EM process, using tools at pain-point places. It is strikingly different from the familiar algorithm-centric architecture of an RDBMS (where human users are relegated to peripheral roles). We believe however that such human-centric solution architectures are necessary to solve "messy" DI problems.

Finally, our work suggests that considering collaborative settings and more expressive UIs is important. It also suggests that in such settings (as well as in crowdsourcing) the problem of effective communications among the participants to share knowledge about the problem/data/tools and to converge (e.g., to a problem definition) remains a major HIL challenge.

4. RELATED WORK

EM has received much attention [4, 6]. Early EM work develops automatic solutions. Subsequent work increasingly engages humans. The work Ajax [7] allows users to declaratively specify a cleaning program (involving EM) and handle runtime exceptions. Other works (e.g., [21, 2]) engage users to perform active learning for EM. A recent growing body of work applies crowdsourcing to EM [23, 24, 3, 18, 17, 19] or studies crowdsourcing [12]. Recent RDBMS-style data cleaning works that involve EM include BigDansing and Wisteria [13, 9].

Our work significantly differs from these works in that we seek to build an EM management system that helps the user with all aspects of the EM process. This raises many HIL challenges not discussed in prior work (see Section 2). Examples include debugging blockers/matchers, how-to guides, tools for pain points, and the need for helping the user gain knowledge about the problem/data/tools. In particular, the solution architecture (for the development stage) in Figure 6 with its heavy human-centric emphasis is very different from the architectures of prior work.

Our work distinguishes the development and production stages. The vast majority of current DI work does not make this distinction, and sometimes it is not clear which stage a particular work (e.g., Ajax) addresses. Most DI works appear to have focused on the production stage. Several works (e.g., Potter Wheel, Trifacta [20, 11]) have considered the development stage, as we do in our work.

A growing body of EM work has engaged human users, especially those using crowdsourcing [23, 24, 3, 18, 17, 19]. But they have considered relatively restrictive UIs, such as those allowing a crowd worker to label a set of tuple pairs at a time. In contrast, our experience here suggests that more expressive UIs may help human users be more effective.

Many EM systems have been developed, both in academia and in industry (see [14] for a detailed discussion). These systems however often do not address the end-to-end EM process. They do not study HIL challenges in depth, as we do, and they do not use human-centric solution architectures such as the one in Figure 6.

Finally, there is a growing body of HIL work for data analytics (e.g., [11, 15, 10, 12, 9], see also this workshop series [1]). As far as we can tell, however, none of the work has addressed HIL challenges in EM in depth, as we do here.

5. CONCLUSIONS & ONGOING WORK

HIL research is increasingly critical for data analytics. In this paper we have discussed HIL challenges in several recent EM projects of ours, and lessons which may be applicable to other problem settings. We also hope that more researchers will investigate EM, as it can be a rich playground for HIL research. We are currently addressing the HIL challenges outlined in this paper, incorporating the solutions into open-sourced systems, and evaluating these systems using real-world users in classes and at organizations.

6. REFERENCES

[1] HILDA http://hilda.io/2016/.

[2] K. Bellare, S. Iyengar, A. G. Parameswaran, and V. Rastogi. Active sampling for entity matching. In *SIGKDD*, 2012.

[3] C. Chai et al. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD*, 2016.

[4] P. Christen. *Data Matching*. Springer, 2012.

[5] S. Das et al. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, 2017.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE TKDE*, 19(1):1–16, 2007.

[7] H. Galhardas et al. Ajax: An extensible data cleaning tool. In *SIGMOD*, 2000.

[8] C. Gokhale et al. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.

[9] D. Haas et al. Wisteria: Nurturing scalable data cleaning infrastructure. *PVLDB*, 8(12):2004–2007, 2015.

[10] J. He et al. Interactive and deterministic data cleaning. In *SIGMOD*, 2016.

[11] J. Heer, J. M. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.

[12] A. Jain, A. D. Sarma, A. G. Parameswaran, and J. Widom. Understanding workers, developing effective tasks, and enhancing marketplace dynamics: A study of a large crowdsourcing marketplace. *PVLDB*, 10(7):829–840, 2017.

[13] Z. Khayyat et al. BigDansing: A system for big data cleansing. In *SIGMOD*, 2015.

[14] P. Konda et al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[15] S. Krishnan, D. Haas, M. J. Franklin, and E. Wu. Towards reliable interactive data cleaning: A user survey and recommendations. In *HILDA*, 2016.

[16] E. LaRose et al. Entity matching using Magellan: Matching drug reference tables. In *AMIA Joint Summits*, 2017.

[17] A. Marcus et al. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[18] A. Marcus and A. Parameswaran. Crowdsourced data management: Industry and academic perspectives. *Found. Trends databases*, 6(1-2):1–161, 2015.

[19] B. Mozafari et al. Scaling up crowd-sourcing to very large datasets: A case for active learning. *PVLDB*, 8(2):125–136, 2014.

[20] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, 2001.

[21] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *SIGKDD*, 2002.

[22] M. Stonebraker et al. Data curation at scale: The Data Tamer system. In *CIDR*, 2013.

[23] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[24] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.