Supercharging Trial-and-Error for Learning Complex Software Applications

Damien Masson
dmasson@uwaterloo.ca
Autodesk Research
Toronto, Ontario, Canada
Cheriton School of Computer Science,
University of Waterloo
Waterloo, Ontario, Canada

Jo Vermeulen
jo.vermeulen@autodesk.com
Autodesk Research
Toronto, Ontario, Canada

George Fitzmaurice
george.fitzmaurice@autodesk.com
Autodesk Research
Toronto, Ontario, Canada

Justin Matejka
justin.matejka@autodesk.com
Autodesk Research
Toronto, Ontario, Canada

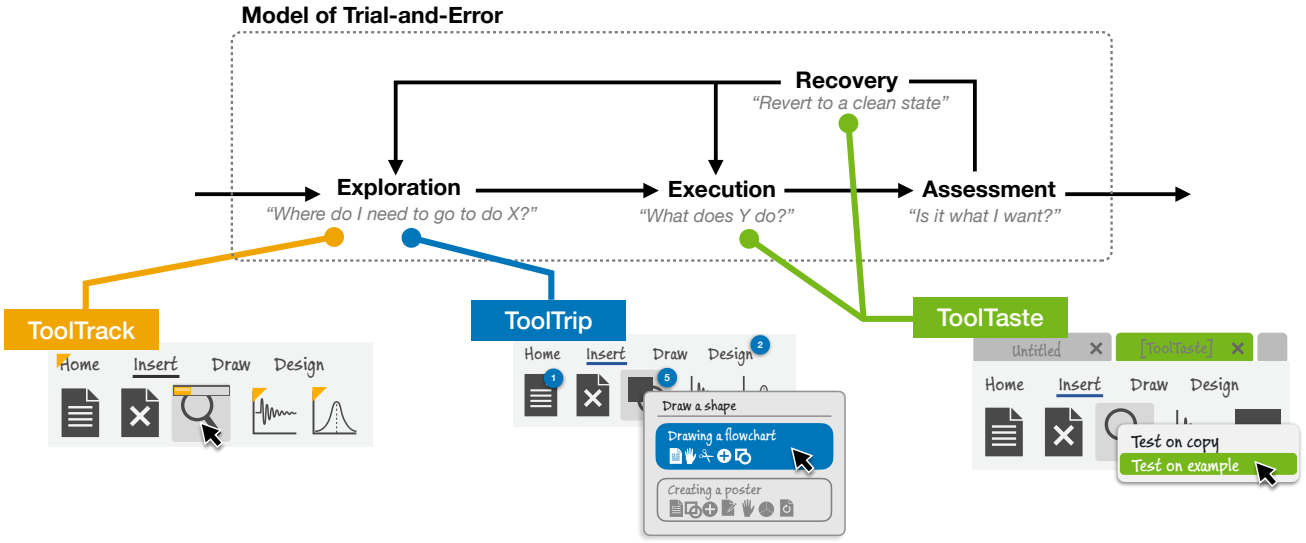

Figure 1: We present a Conceptual Model for trial-and-error and three techniques that improve support for trial-and-error in complex software at the Exploration, Execution and Recovery phases: *ToolTrack*, *ToolTrip* and *ToolTaste*.

ABSTRACT

Despite an abundance of carefully-crafted tutorials, trial-and-error remains many people's preferred way to learn complex software. Yet, approaches to facilitate trial-and-error (such as tooltips) have evolved very little since the 1980s. While existing mechanisms work well for simple software, they scale poorly to large feature-rich applications. In this paper, we explore new techniques to support trial-and-error in complex applications. We identify key benefits and challenges of trial-and-error, and introduce a framework with a conceptual model and design space. Using this framework, we developed three techniques: *ToolTrack* to keep track of trial-and-error progress; *ToolTrip* to go beyond trial-and-error of single commands by highlighting related commands that are frequently used together; and *ToolTaste* to quickly and safely try commands. We demonstrate how these techniques facilitate trial-and-error, as illustrated through a proof-of-concept implementation in the CAD software Fusion 360. We conclude by discussing possible scenarios and outline directions for future research on trial-and-error.

CHI '22, April 29-May 5, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-9157-3/22/04…$15.00
https://doi.org/10.1145/3491102.3501895

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; **HCI theory, concepts and models**; *Graphical user interfaces.*

KEYWORDS

trial-and-error, learning by exploration, software learning, technique, conceptual model, design space

ACM Reference Format:
Damien Masson, Jo Vermeulen, George Fitzmaurice, and Justin Matejka. 2022. Supercharging Trial-and-Error for Learning Complex Software Applications. In *CHI Conference on Human Factors in Computing Systems (CHI '22), April 29-May 5, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3491102.3501895

1 INTRODUCTION

Discoverability and explorability were key advantages of early graphical and direct manipulation user interfaces [38]. By relying on recognition rather than recall, these interfaces accelerated learning new software because people did not have to learn and memorize complex syntax. Instead, users had access to continuous visual representations of the objects of interest and the possible actions through icons and menus (i.e., visibility [29]), continuous feedback, and reversible actions, all of which facilitated exploration. These ideas have left a lasting mark on user interface design ever since. Indeed, people nowadays still rely on – and prefer – to use trial-and-error to learn how to use unfamiliar software or explore new functionality, rather than reading documentation [1, 17]. Yet, the way in which software facilitates trial-and-error has evolved very little since the 1980s, mostly relying on text tooltips and recognizable icons and labels. While these approaches work well for relatively simple applications, they are less effective for complex feature-rich software applications such as Adobe Illustrator or Autodesk AutoCAD [1, 24, 30]. Hence, when users get stuck during trial-and-error, their only recourse is to seek explicit help by consulting official documentation, asking others, or searching the web.

Early on, researchers noted the tendency to learn-by-doing and formulated guidelines to support exploration of the interface [7, 8, 32]. However, these guidelines are often hard to follow in practice, and it is unclear how modern feature-rich applications could implement them effectively. For example, with ever evolving software and a growing number of commands, it is challenging to "keep the number of possible operations small", or "make the possible operations distinguishable" and "continuously visible" [7, 32].

Research in software learning often does not specifically target trial-and-error; instead, the community tends to focus on other aspects of learnability (e.g., surfacing contextually relevant tutorials). This may be explained by the tendency to see trial-and-error as an unproductive approach compared to explicit help such as tutorials, as trial-and-error can lead to learning suboptimal solutions and can result in users asymptoting at mediocre performance levels [6]. As a result, explicit support for trial-and-error remains largely unexplored. However, given people's natural tendency to rely on trial-and-error, we believe there is an exciting opportunity for more in-depth research into how we can improve trial-and-error, particularly for large and feature-rich applications.

In this paper, we make the following contributions:

- **A Framework for Trial-and-Error**: We review the literature on trial-and-error and extract four *benefits* and four *challenges* (Section 4), which we develop into a *conceptual model* (Section 5.1) and *design space* (Section 5.2) of trial-and-error. This framework can be used in a generative way to compare existing trial-and-error approaches and identify opportunities for future research.
- **Three Trial-and-Error Techniques**: We introduce three techniques and demonstrate how these facilitate trial-and-error for complex software, linking to challenges in the conceptual model and gaps in the design space:
  - *ToolTrack* allows users to track their trial-and-error progress and quickly locate commands of interest (Section 6.1);
  - *ToolTrip* goes beyond trial-and-error of single commands by showing related commands that are often combined (Section 6.2);
  - *ToolTaste* lets users rapidly and safely test and experiment with commands (Section 6.3).
- **Fusion 360 Implementation**: We implemented the above techniques in the CAD software Fusion 360 (Section 7).

2 BACKGROUND

2.1 What is Trial-and-Error?

Broadly speaking, *trial-and-error* consists of trying different approaches to solve a problem (trials), discarding failures (errors), and repeating until one is successful. *Trial-and-error* is a fundamental method of problem-solving, and has applications in various domains (e.g., finding new drugs [15], solving puzzles [39], etc.). In this paper, we refer to *trial-and-error* in the context of software learning [1, 30, 36], also known as *learning by exploration* [7, 8, 32], *exploratory learning* [36], and *self-directed exploration* [17, 24].

In terms of when trial-and-error is used, observational studies [8, 17, 30, 36] have found it to be used in three main contexts:

As a first approach: While some users might prefer reading documentation first, self-guided exploration was found to be used more than half the time when attempting a new problem [1, 17].

In combination: Help resources and trial-and-error are often combined to find new aspects of the interface to explore or to disambiguate help instructions that appear unclear [36]. For example, when users get stuck using trial-and-error, they may look for a video online, scrub through the video to identify the relevant commands, and then finish the task through trial-and-error in the application, ignoring most of the video [17].

Task-free exploration: Trial-and-error is also often the strategy of choice when users do not have a specific goal in mind other than learning how to use a new tool [8]. This last use case arises when users try out new software and want to assess its capabilities, without having a specific task in mind.

2.2 Why Do People Prefer Trial-and-Error?

While there is no single answer to explain why people prefer trial-and-error, several reasons are mentioned in the literature that can explain this preference.

The Paradox of the Active User: This paradox refers to the common observation that users refrain from reading manuals and instead start to immediately use the software [5]. Carroll and Rosson identify two biases displayed by software users that help explain this phenomenon: *production bias* and *assimilation bias*. Production bias refers to the fact that throughput is the paramount goal of users. They have little motivation to learn about the system, and will most likely ignore training material or manuals. Assimilation bias means that users rely on what they already know, even when faced with

completely new problems. Numerous studies have confirmed these biases and found them to be equally important – even exacerbated – with modern and feature-rich software [1, 17, 24, 30]. Users have an "illusion of progress" when using trial-and-error [30], and perceive any tasks that do not directly contribute to the completion of their goal as a waste of time. Moreover, Rieman et al. found that users often feel time pressured when having to accomplish a new task [36]. As a result, users tend to start problem-solving without external help, as they believe that to be the most efficient strategy.

Low threshold: In many ways, trial-and-error is easier to use and get started with than alternatives such as reading manuals or following tutorials. First, users can use trial-and-error without having any knowledge of the applications' *vocabulary*, i.e., terms that have a specific meaning in the application's context, such as "layers" in Photoshop or "headings" in Microsoft Word [30]. In contrast, to seek help, users will need to be familiar with the applications' vocabulary and be able to articulate a search query. In fact, vocabulary mismatch is the leading cause of unsuccessful attempts at finding help [17]. Second, trial-and-error makes use of arguably the most minimalist and concise manual: the user interface. User interface elements are often arranged in ways to optimize for fast scanning to quickly find the item of interest. In contrast, manuals and tutorials can be overwhelming due to large amounts of textual information, while videos can be difficult to scan to find the segment of interest. Thus, these help resources are difficult to skim and users might be faster by exploring the interface [30].

Self-reliance: Users tend to over-estimate their capabilities and trial-and-error's usefulness [1, 30]. Users are constantly faced with user interfaces. Additionally, these interfaces are made to look consistent, sharing the same affordances (e.g., the visual appearance of buttons), similar icons, and, sometimes, the same functionality (e.g., when transitioning between tools within the same application domain). Thus, users are led to believe that they can use software applications without any help [36]. And for the most part, they are right; about 50% of trial-and-error episodes are successful [30]. Even if the outcome of a trial-and-error session is negative, users are not particularly frustrated by it [17] and they easily forget failures, further reinforcing their feeling that they "can do it themselves".

Just-in-time learning: Rieman et al. found that "users often prefer to postpone their learning until driven to it by real tasks" [36]. Trial-and-error has the advantage of allowing users to explore only what is absolutely necessary to accomplish their task. In contrast, a tutorial would often present different possible parameters and capabilities of a command on abstract tasks or toy problems. Thus, by using trial-and-error, users are introduced to new functionality just-in-time to accomplish a real task.

2.3 How Software Facilitates Trial-and-Error

Because of the strong success of "point-and-click" WIMP interfaces and their advantages for ease of learning [42], graphical user interfaces evolved to further facilitate and encourage exploring the interface – and as a result, to support aspects of trial-and-error. Polson and Lewis [32], and Draper and Barton [8] proposed guidelines to support exploratory behaviours, which were later compiled into six guidelines by de Mul and van Oostendorp [7]. Nowadays, software commonly follows most of these recommendations:

(1) *"Keep the number of possible operations small at any given time"*
(2) *"Make the possible operations distinguishable"*
(3) *"Make clear what the consequences of every action will be"*
(4) *"Make the effects of actions visible once they have been executed"*
(5) *"Show the last actions performed by the user"*
(6) *"Make actions easily undoable to make it safe to experiment"*

While modern feature-rich applications like Microsoft Word or Adobe Photoshop implement most of these guidelines, there are still barriers for trial-and-error. For example, Word hides less common commands in its "Ribbon" interface, groups related commands, and only shows particular ribbons depending on the situation (e.g., the "Picture Format" ribbon pops up when a picture is selected). However, while this helps with keeping the number of possible operations small, approaches that hide much of an application's functionality can make it harder to find and explore less popular commands. Adobe Photoshop similarly groups related commands in its Tools Palette, but this can complicate finding a desired tool that is currently not visible. As the number of available commands increases, it also becomes more difficult to distinguish visually similar icons. While tooltips help users to predict the effect of a particular operation, they often only provide short text descriptions with limited information, which has spurred research into more detailed tooltips with video or dynamic previews [13, 40]. Lastly, most software offers "undo" and "redo" functionality, with some applications (like Photoshop) providing a higher level of granularity with an advanced history panel or timeline of past actions. Nevertheless, it remains difficult to safely test and compare the effects of multiple commands, without accidentally losing data [24].

3 RELATED WORK

Despite extensive literature studying trial-and-error [1, 7, 8, 24, 30, 32, 36], and techniques targeting specific aspects of software learnability [14], most systems to date have not specifically targeted trial-and-error. Yet, some of these approaches offer features and opportunities that could benefit trial-and-error.

3.1 Guiding Exploration

One way to help users with trial-and-error when a software application contains a large amount of commands (sometimes in the thousands) is to guide their exploration towards only a subset of those commands. Several systems propose to gather information from the user and from the larger community to guide exploration. This can be done at a micro-level, through Scented Widgets [43], or at a macro-level, by overlaying a heatmap of the most frequently used commands [26]. Alternatively, instead of overlaying information, the interface can adapt to make certain commands more prominent. For example, menus can be organized depending on selection frequency [9, 37] and elements can be gradually faded in [10]. In the same vein, Carroll and Carrithers used the "training wheels" metaphor to propose interfaces that show only a subset of the commands [4]. The interface then unlocks additional features as users get more familiar with it.

Our work differs in that we specifically look at exploration in the context of trial-and-error; while directing users' to the most

frequently used commands might be beneficial to discover commands, it is arguably not particularly helpful during a trial-and-error episode, in which users have a task in mind and try to accomplish it as quickly as possible. Instead, we look at other cues such as marking whether commands have already been explored previously (Section 6.1), to draw users attention to missed but possibly relevant commands [30] and prevent them from repeating mistakes by selecting the wrong command [24].

3.2 Recommending Actions

Other techniques focused on recommending the users' next actions to help them discover new commands. For example, the OWL system [20, 21] and CommunityCommands [27] provide command-level recommendations; the former models command usage and then compare it across a pool of users while the latter applies collaborative filtering algorithms to present command recommendations to users. DiscoverySpace [11] explored task-level recommendations in Photoshop to help novices accomplish tasks by harvesting and suggesting one-click action macros. Together, these recommendations systems have shown promising results to help novice users learn and use complex software.

We draw inspiration from recommender systems and these prior techniques to propose workflow-level suggestions for trial-and-error. Specifically, we recommend workflows (i.e., sequences of commands) to facilitate trial-and-error by illustrating how commands can be used, boosting serendipitous discovery, and prompting users with alternative approaches (Section 6.2).

3.3 Experimenting and Exploring Alternatives

One of trial-and-error's key principles is to repeatedly experiment and try out different operations, but this process can be tedious and slow in complex software applications. Several systems have aimed to facilitate part of the experimentation phase. For example, Side Views [40] offers evolved preview capabilities allowing users to see and explore multiple variations of the parameters associated with a command. Similarly, Subjunctive Interfaces [22, 23] and Parallel Paths [41] let users work with multiple copies of the data and explore different parameters in parallel. Lastly, several more sophisticated "undo" systems have been proposed (see Nancel & Cockburn [28] for an overview) that allow users to selectively change past actions to explore alternatives without having to start from scratch [34, 35]. Alternatively, Lafreniere et al. proposed a set of command disambiguation techniques [19]: Did-You-Mean and Or-do allow users to undo the recent command and replace it by another one, transferring parameters if possible.

Inspired by these approaches, we go beyond prior undo systems by allowing safe experimentation through the use of an explicit *sandbox* mode for trial-and-error, in which users can easily discard their results or apply them to their current document (Section 6.3).

4 PROPERTIES OF TRIAL-AND-ERROR

To better understand trial-and-error, we analyzed empirical studies of people's behaviour when learning new software applications. We reviewed and extracted sentences qualifying trial-and-error from eight observational and diary studies that covered a broad range of user profiles (from all ages [24] and with different levels of technical expertise [7, 17, 36]) and application domains (drawing [8], email [7], marketing [1, 30], 3D design [17], note-taking [24], photo editing [19]), as shown in Table 1.

Property	[8]	[7]	[36]	[1]	[30]	[17]	[24]	[19]
B1: High success rate	✓	✓	✓	✓	✓		✓	
B2: Feeling of progress					✓	✓		
B3: No context switching			✓					
B4: Discover commands			✓	✓				
C1: Cannot find commands	✓			✓	✓	✓	✓	✓
C2: Cannot operate commands	✓				✓			
C3: Suboptimal solutions			✓	✓	✓			
C4: Costly recovery							✓	

Table 1: Breakdown of the eight properties of trial-and-error and which study observed and mentioned them.

Our analysis resulted in eight properties of trial-and-error that were observed, often in several of the studies. We categorized these into four benefits (B1–B4) and four challenges (C1–C4). Table 1 shows the breakdown of the properties and the papers they were mentioned in. Next, we briefly discuss each of these properties.

4.1 Benefits

B1: High success rate – Trial-and-error often performs equally, if not better [7, 24, 30] than using help. For example, Novick et al. found higher success rate and shorter completion times when using trial-and-error compared to help [30]. This is especially important considering that users tend to use trial-and-error more often than other problem-solving strategy [1, 17], or in combination with other strategies like disambiguating information from manuals [36].

B2: Feeling of progress – Trial-and-error allows users to directly work toward achieving their goal [36], giving them a feeling of progress [30]. From a user-perspective, reading training materials and following tutorials with artificial tasks might feel like a waste of time. Instead, when using trial-and-error, users set their own tasks, and thus have the feeling of getting closer to their objective. In fact, this feeling leads users to spend more time on trial-and-error than help, even if using help may be faster [30].

B3: No context switching – By definition, trial-and-error happens in the software, which saves users from going back and forth between an external help source (e.g., forums, YouTube, web searches) and the application. In contrast, help resources – even when integrated directly in the application – will reduce the space allocated for the task, create an interruption, and increase cognitive load [12, 36].

B4: Discover commands along the way – Even in a successful trial-and-error session, finding the right command often requires multiple unsuccessful attempts involving a variety of commands. These unsuccessful attempts might feel like a waste of time. However, in the long run, they help to get the user familiar with the interface. When faced with a new task, a user might recall a command that they "stumbled onto by accident" during trial-and-error [1, 36].

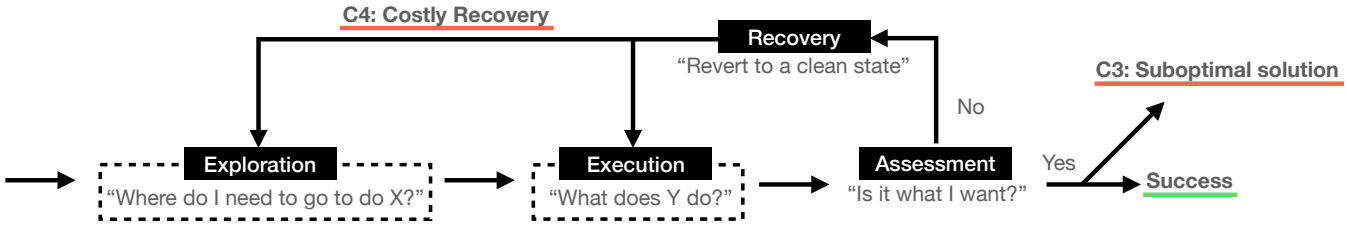

Figure 2: Our conceptual model of trial-and-error. References to the challenges presented in Section 4.2 are underlined in red. The Exploration and Execution phases are further detailed in Figure 3 and Figure 4 respectively.

4.2 Challenges

C1: Cannot find commands – A trial-and-error session might reach a dead-end once users run out of cues to follow [30]. Users tend to select menus and commands which appear to lead the most quickly to their goal (i.e., they use a label-following heuristic [32]). Previous studies have identified various causes to these dead-ends: the relevant command is hidden or disabled [17]; the affordances are poor or misleading [19]; the user's vocabulary differs from the one used by the software [1, 17, 30]; or the user missed the command because they did not look in the right place [24, 30]. Once stuck, the only recourse for users is to resort to a different problem-solving strategy such as looking for help [1, 17].

C2: Cannot operate commands – Users can face difficulties understanding how to navigate a specific command. Often, this is due to an incomplete or mistaken mental model of the application (e.g., users do not understand that objects are locked or grouped) [1]. At best, users will abandon the command and try a different – often less efficient – approach, involving a different command; users are rarely completely stuck when using trial-and-error [8]. At worst, it can lead to roadblocks in a trial-and-error session as users might be unable to assess if the command is relevant to their task.

C3: Suboptimal solutions – Novick et al. found that trial-and-error episodes sometimes resulted in unconventional ways of completing a task [30]. Indeed, when feeling under time pressure and having little desire to learn the software, people may stick to the first solution that they stumble across, even if it is inefficient. While these approaches appear successful, they are often slower or bad practices which might be problematic later on [30].

C4: Costly recovery – Reverting to a clean state after an unsuccessful attempt is crucial for trial-and-error. Yet, this recovery can be costly. For example, Mahmud et al. [24] found that older adults often tried to undo actions that impacted the interface (e.g., selecting a different brush) but those are not typically part of the undo stack. As result, they undid the wrong thing and accidentally lost data.

5 FRAMEWORK

We propose a framework including a conceptual model of trial-and-error and a design space of techniques that provide support for trial-and-error. The goal of this framework is threefold: (1) improve our understanding of trial-and-error; (2) identify how previous work impacts trial-and-error; and (3) reveal opportunities to improve software applications' support for trial-and-error.

5.1 Conceptual Model of Trial-and-Error

To better understand when users face the challenges described in Section 4.2, and to find solutions to those challenges, we propose a conceptual model of trial-and-error (Figure 2). We designed this model based on our review of the literature on users' behaviours when using trial-and-error. As such, we define a complete episode as four consecutive phases that are repeated until achieving success. We describe these phases and applications of our model in the following sections.

5.1.1 *Phases.*

Exploration: Given a set of commands, in the *Exploration* phase, users will attempt to find commands that are likely to achieve the desired result. The exact exploration strategy differs between users [24], and their experience with the application itself as well as previous applications [30]. On one extreme, this search can appear random and exhaustive [30]. On the other extreme, the exploration can be targeted when users have partial knowledge about the application [30]. In both cases, this strategy is often described as a *label-following heuristic* [32], as mentioned before (see Section 4.2). Usually, reviewing a potential command is done in multiple steps (see Figure 3) [24]: first, users will search based on features easily accessible at a glance (e.g., icon, label, position). If a command's meaning remains unclear after this step, some users will use a more "costly" filtering, if available (e.g., reviewing the description in the tooltip of commands). If a candidate command was identified, this step leads to the *Execution* phase. Otherwise, the user's trial-and-error episode might end here because they could not identify a relevant command (C1).

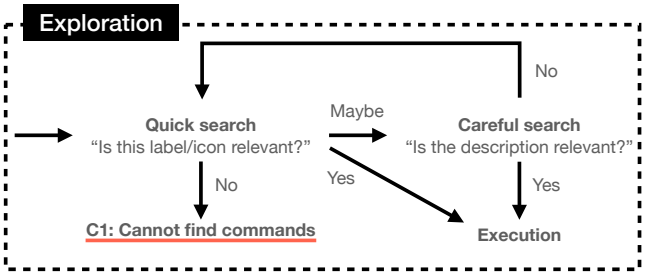

Figure 3: The exploration phase in the conceptual model.

Execution: Once a candidate command is identified, the next step is to try it. At this stage, Draper and Barton differentiate between two kinds of user goals [8]: "experimenting with a command

to learn what it does" and"looking for a command to achieve a specific goal". In the first case, users experiment with the command and explore the different parameters and options, rapidly testing different variations (even if they are not useful to them) by skipping *Assessment* and directly going to *Recovery*. In the second case, users are more targeted and only explore the relevant parameters and options to get their task done. Once completed, they will proceed to *Assessment*. In both cases, users may face a breakdown in the trial-and-error process if they cannot operate the command (C2).

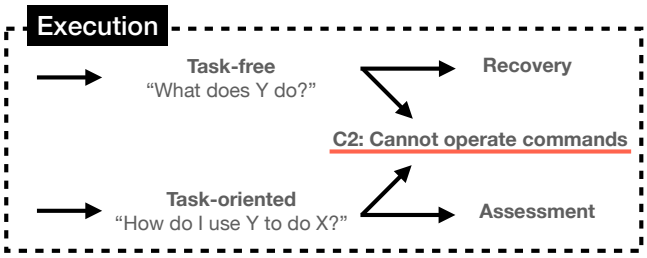

Figure 4: Conceptual model of the execution phase.

Assessment: After executing the command, users will have to decide if they are satisfied with the result. In practice, this phase often boils down to checking if the result *appears* correct. Consequently, even if this result is a workaround and differs from what an expert would have done (C3) [30], it might satisfy users and they will most likely stick with it (i.e., assimilation bias [5]). If users are satisfied, then the trial-and-error episode is complete and successful. Otherwise, users will have to go to the *Recovery* phase.

Recovery: This last phase occurs in preparation of a new trial when a clean state is necessary to allow for another attempt. This can be done by manually reverting the changes, or by using the built-in undo mechanism. In both cases, this recovery might be tedious and error-prone [24] (C4). Once the prior state has been recovered, users might switch to a different help strategy [1], go back to *Execution* if they want to further experiment with the command, or return to *Exploration* to look for a different command [24].

5.1.2 *Generalization of the Model.*

While we presented our conceptual model using *commands* for clarity, these only represent a subset of the operations that users commonly learn through trial-and-error. In addition to commands, prior studies also observed users using trial-and-error to figure out (1) the *user interface* (e.g., open or close a view, change the tool currently selected, customize the interface), (2) *parameters* associated with a command, and (3) *workflows* (e.g., find the most efficient sequence of commands to accomplish a particular task) [1, 24, 30]. We designed our model to be general enough to include these variations. One can read the previous explanation and replace *command* by either *user interface*, *parameter* or *workflow* as needed.

Our model is also recursive: a trial-and-error episode can contain sub-trial-and-error episodes. For example, when using trial-and-error to experiment with different workflows, a user might be faced with new commands. The user can then momentarily pause their exploration of different workflows to start a new trial-and-error episode aiming to understand the new command. Once the sub-trial-and-error episode is over, users will return to the main trial-and-error session – i.e., exploring workflows. Similarly, users can fall into "undirected diversion" in which they discovered an interesting feature and decide to take a break from their trial-and-error session to investigate the new feature through task-free exploration.

5.1.3 *Using the Model.*

This conceptual model can serve as a generative tool to inspire new solutions to support trial-and-error and to classify and understand existing approaches. Here, we provide a few examples of how the model can help to clarify support for trial-and-error in existing techniques and how it can inspire new techniques.

For example, adding "Patina" [26] to a software application provides an additional filtering step during the *Exploration* phase, in which the user can look at the frequency of use for each command to assess whether it is relevant or not. Similarly, adding the "Ordo" feature from Lafreniere et al. [19] would create a direct path between the *Recovery* and *Assessment* phases – users can recover from an undesired command by replacing it with another, with the parameters transferred, essentially skipping the *Execution* phase.

In terms of inspiring new techniques, as shown in Figure 1 and as we will describe in Section 6.3, *ToolTaste* provides additional support for *Execution* by always allowing execution of commands (addressing challenge C2), even if that command is disabled in the current situation, and for *Recovery* by allowing users to more easily compare different variations of a command and set their own restoration points (addressing challenge C4).

5.1.4 *Related Models.*

Don Norman identified two "gulfs" that people have to bridge to use something: the *Gulf of Execution* and the *Gulf of Evaluation* [29]. Using the *Seven Stages of Action* as described by Norman, one can model part of the trial-and-error process (with the stages of execution corresponding to the *Exploration* and *Execution* phases, and the stages of evaluation corresponding to the *Assessment* phase). However, this representation does not consider the *Recovery* phase and the non-linear nature of trial-and-error. Our model differs by being more specific and modelling trial-and-error's core components such as the Execution-Assessment-Recovery loop or the *Exploration* phase. This allows designers to identify where difficulties appear during the trial-and-error process, and helps identify gaps or new paths that could be created (as discussed in Section 5.1.3).

Inspired by animal food foraging strategies, Pirolli and Card proposed information foraging theory [31], which explains how "information scent" cues can help people to navigate information and make decisions. This theory can help understand trial-and-error; in this case, users are interested in accomplishing a task with the software at hand. However, while information foraging tries to explain why and when users would change site (or, in our context, try a different tool) to maximize their chances of attaining their goal, our model describes the general pattern followed by users during a trial-and-error session. Thus, information foraging theory is complementary to and can be combined with our model to better understand users' choices (e.g., why a tool was chosen or abandoned in favour of another), and to consider providing additional information scent cues to help people in their trial-and-error process. The techniques we will later introduce (Section 6)

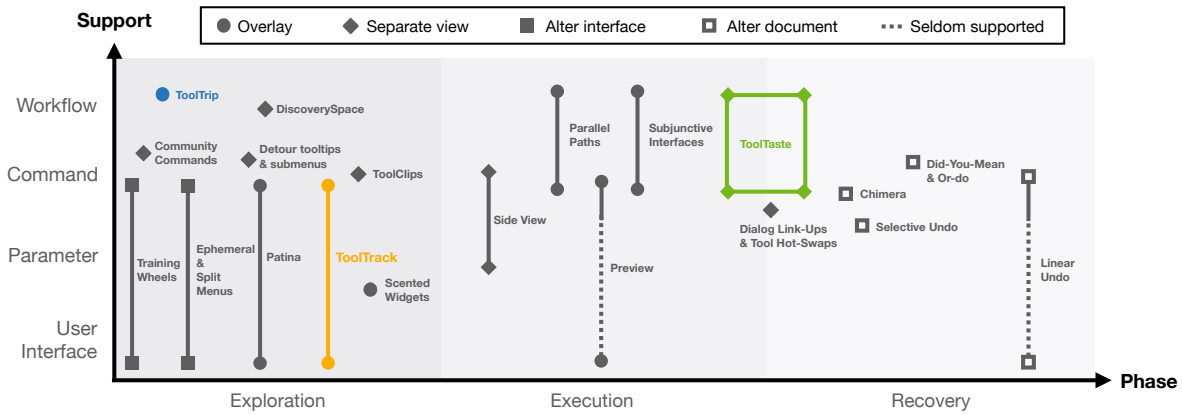

Figure 5: Design space of support for trial-and-error.

provide further information scent cues in addition to the existing tooltips and icons to help direct users to tools they have not explored yet (*ToolTrack*) or are likely to be of interest for the task they want to achieve (*ToolTrip*), or can make it easier to quickly and safely try something out (*ToolTaste*).

5.2 Design Space of Support for Trial-and-Error

From a review of the literature and our conceptual model, we propose a design space for tools aiming to support trial-and-error (Figure 5). Designing support for trial-and-error implies choosing what to support, when the support should intervene, and how to present that support to the user. Throughout this section, we present the design space dimensions and illustrate them using examples from prior research. Note that we use the word "document" to refer to the data being manipulated. Thus, "document" will correspond to a text document in Microsoft Word, a drawing in Photoshop, a 3D model in Fusion 360, etc.

5.2.1 *Types of support.*

Trial-and-error tasks in complex software pertain to various elements of the interface. Below, we list four aspects of complex software frequently targeted by trial-and-error.

- **User Interface**, which corresponds to actions with a scope limited to the interface without impact on the document. For example, hiding a side-panel or changing the currently selected tools.
- **Parameter**, which corresponds to the different values associated with a command. For example, the rectangle's width when using a command to draw a rectangle.
- **Command**, which corresponds to operations to modify the document. For example, copy/paste, or application specific-commands (e.g., centering selected text in Microsoft Word).
- **Workflow**, which corresponds to a sequence of commands. A user might be familiar with the commands and how they work, but may lack knowledge on how to combine them to accomplish a particular task. Alternatively, a user might be looking for a more efficient workflow to achieve the desired result.

Most systems offer support at a command-level, either by directing users' attention to specific commands [26] or suggesting commands to try [27]. Support for parameters has also been thoroughly explored, by offering ways to visualize the effect of parameter variations [40]. However, to the best of our knowledge, few systems offer support at the workflow-level. Notable exceptions are DiscoverySpace [11], which suggests action macros, and Subjunctive Interfaces [22, 23], which allows the exploration of multiple variations of the document.

5.2.2 *Time of Intervention.*

Support for trial-and-error can intervene at different stages of the exploration. We consider three stages, extracted from our conceptual model, in which an intervention is possible: **Exploration**, **Execution**, and **Recovery**. Details about each phase are presented in Section 5.1. We excluded *Assessment* from our list because this quick decision phase offers little opportunity for an intervention.

Systems such as Patina [26] (which shows frequently used commands through an overlay), and ToolClips [13] (which augments tooltips with video clips) intervene during the *Exploration* phase. In contrast, typical preview systems (e.g., a thumbnail with a preview of the filter in GIMP) and more advanced previews such as Side Views [40] allow users to understand a command and each parameter's effect during the *Execution* phase. Lastly, the "Did-You-Mean" and "Or-do" systems proposed by Lafreniere et al. [19] intervene during the *Recovery* phase.

5.2.3 *Types of Presentation.*

From previous work, we identified fours ways of presenting the support to users, categorized from least to most obtrusive:

- **Altering the document**, which is the simplest form of presentation but also the least obtrusive; users see the modification because their document has changed.
- **Altering the interface**, also referred to as an adaptive UI, which consists of changing, hiding or moving elements of the interface.
- **Overlaying,** which consists of showing the information on top of the interface or the document, without obstructing the view.
- **In a separate view**, which opens an additional view with the information. This presentation mode is the most flexible approach but also the most obtrusive one as it reduces the space allocated by the software application.

This dimension has been extensively explored by existing systems. For example, "undo" systems typically only alter the document while Side Views [40] and CommunityCommands [27] display information through separate views. Patina [26] and ScentedWidgets [43] use overlays to show information gathered from the community. Finally, systems such as Training Wheels [4] alter the interface to restrict the number of commands.

6 SUPERCHARGING TRIAL-AND-ERROR

In this section, we show how our framework can be used to design new features that tackle some of the challenges identified in Section 4.2. We reviewed previous work on systems that support trial-and-error in Figure 5. Note that this excludes approaches that do not intervene during trial-and-error, such as high-guidance systems like stencil-based tutorials [16], and other systems that require users to articulate search queries or read manuals [18, 25]. From this analysis, we found that three issues remained poorly covered:

(1) users cannot keep track of their progress during a trial-and-error episode;
(2) experimenting with commands and recovering from workflows is difficult; and
(3) exploring at a workflow-level is mostly unsupported.

Throughout this section, we describe the design and rationale behind *ToolTrack*, *ToolTrip* and *ToolTaste* – three features compatible with complex applications that tackle the aforementioned issues.

6.1 ToolTrack: Show Coverage and Track Progress

In large complex software applications, the *Exploration* phase can be tedious; users often lose track of their progress and end up retrying commands that they discarded earlier [24], or discard commands too early, before exploring relevant command parameters [30] (C1). Drawing inspiration from previous work helping with the exploration such as Patina [26], we propose *ToolTrack* (Figure 6) to overlay information about one's prior exploration on top of commands. We define three levels of coverage of a command:

- *level 0*: The command has never been used;
- *level 1*: The command's tooltip has been opened at least once;
- *level 2*: The command has been executed at least once.

At the last level, we calculate a finer granularity of coverage by calculating the percentage of parameters' explored, e.g., a command with five parameters will be considered fully explored once all five parameters have been modified at least once. *ToolTrack* then modifies the interface to show different feedback depending on how well a command has been explored. A level-0 command will be shown with a top-left yellow corner; a level-1 command will be shown with a small top-left yellow corner; a level-2 command will be unaltered, but will show a progress bar on hover to present the percentage of the command's parameters explored.

ToolTrack was designed to be discreet as to not hinder typical use, while also providing useful cues in trial-and-error episodes. As such, a user will be able to quickly locate unexplored commands by noticing the commands that display a yellow triangle. This helps users keep track of their progress (B2) as well as locate potentially relevant commands (C1). Similar to Patina [26], *ToolTrack* helps users to find previously used commands by looking for those that do not display a yellow corner. Finally, *ToolTrack*'s last objective is to motivate further exploration of commands and parameters in the hope of revealing alternative solutions (C3).

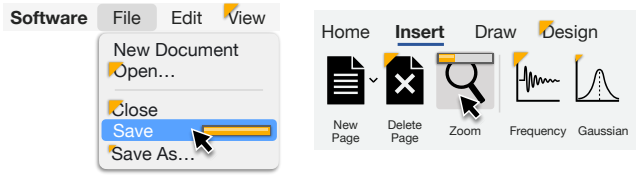

Figure 6: ToolTrack shows unexplored commands with a yellow triangle, and for commands that have been used before, it shows a progress bar indicating how deeply that command has been explored.

6.2 ToolTrip: Exploration at Task-Level

Users' goals with complex software are often high-level (e.g., "creating a roof") and will require the combination of different commands. Yet, common cues in complex software only inform users at a command level, meaning that users have to figure out themselves how to arrange commands to achieve their goal. This lack of support for trial-and-error at the task or workflow level is apparent when looking at the design space (Figure 5). While DiscoverySpace [11] provides workflow support by suggesting and allowing users to execute action macros (i.e., sequences of operations), it does not allow users to pick and choose commands from these sequences, nor learn from them. Thus, we extend this idea further through *ToolTrip* (Figure 7), which let users go on "trips" in which all the commands necessary to reach the end are presented. Trips are presented with a title and a brief description of the end goal. Following a trip is done with as little guidance as possible to let users trial-and-error. Indeed, considering the infinite number of arrangements of commands and workflows, it is unlikely that users will find the exact *ToolTrip* that corresponds to their task. However, part of a given workflow might still be useful to their task. In the meantime, being able to rapidly examine a large amount of possible ways to go about a problem directly in the application might lead them to find better solutions (C3) or discover relevant commands (C1). Finally, *ToolTrip* allows users to examine possible usage of a command when combined with other commands. Moreover, these suggested *ToolTrips* can be personalized based on the user's recent history.

Users are presented with different *ToolTrips* with a brief description of the outcome of the trip (i.e., the effect of executing the full sequence of commands). Once a *trip* is selected, commands in the trip will display a coloured badge (❶, ❷, ❺ in Figure 7). A new view will be pinned in the interface showing the name of currently selected trip, the icons of the commands composing the trip, and a button to stop following this trip. By hovering over an icon, the corresponding command is shown to the user, possibly changing the currently selected view in the user interface (e.g., a tab in the application's ribbon) if needed to show that command.

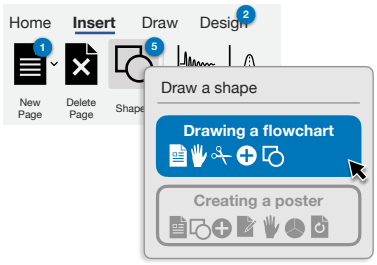

Figure 7: ToolTrip offers workflows that contain a particular command under the mouse cursor, highlighting other commands in that workflow with numbered badges.

6.3 ToolTaste: Rapid Testing of Commands

Experimentation is key to trial-and-error, yet, trying commands is difficult in complex software. First, commands often have preconditions, which, if not met, will make the command unusable or appear to have no effect (C2) (e.g., making text bold in Microsoft Word requires one to first select text). Second, experiments have to be done on the document at hand, possibly resulting in an accidental loss of data when trying to recover (C4). Recovery is especially difficult when exploring multiple "depths" of commands as most software offers little control over the granularity of the "Undo" system (see Figure 5).

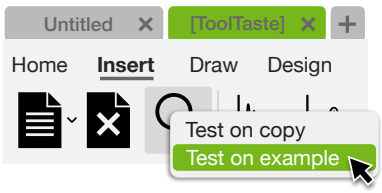

Figure 8: ToolTaste allows users to test any command, even if it is currently disabled – either on the current document or on an example that has been curated to work with that command.

ToolTaste allows users to quickly and safely test a command in order to assess its relevance for the task at-hand. Users can right-click on any command – even disabled ones – and select "Test on example" or "Test on copy". Users are then moved to a different view with an example-project or a copy of the current document loaded so that users can experiment with the command independently from their main project. At any time, users can reset the document (to test a new set of parameters) and go back-and-forth between the main view and *ToolTaste* to compare changes. Once done exploring, the changes can be merged with the main document, or discarded.

ToolTaste differs from traditional recovery systems in that users set their own "restoration points" on a specific command, which has two advantages. First, users have control over the recovery stack, allowing them to recover from a long chain of commands. Second, because *ToolTaste* requires an explicit action from the user, we know *when* a trial-and-error episode starts and for *which* command. In *ToolTaste*, we use this opportunity to offer example-projects tailored to each command, but other systems could leverage this information to show help related to the command (e.g., Ambient Help [25]).

7 PROOF-OF-CONCEPT IMPLEMENTATION IN FUSION 360

We implemented *ToolTrack*, *ToolTaste* and *ToolTrip* in the Autodesk Fusion 360 Computer Aided Design (CAD) tool. Fusion 360 is a feature-rich software application used by professional and recreational users to create sketches of 3D models and turn them into 3D printable objects. We chose Fusion 360 as it is difficult to learn [17], but also because it has a large community of users. We directly modified Fusion 360's source code to implement these three techniques, using C++ and Qt [1], as discussed below.

7.1 Interaction and Visualization

We implemented *ToolTrack* to show how much a command has been explored, in addition to existing information such as icons and labels (Figure 9A). We modified all the buttons in the ribbons and menus of Fusion 360 to overlay them with *ToolTracks*. A command that has never been explored will have a yellow corner, this corner decreases in size if the command' tooltip has been looked at, and will completely disappear once the command has been executed once. Hovering over the button or menu will show a progress bar indicating how many parameters associated with this command have been explored.

We augmented Fusion 360's tooltips by adding: a progress bar showing a *ToolTrack* (i.e., how much of the command has been explored), frequent next commands as extracted from *ToolTrips*, and three popular *ToolTrips* that involve the command being hovered (Figure 9B). Each *ToolTrip* has a distinct colour (either red, green or blue), a title, and the ordered sequence of unique icons for each command composing the trip (limited to 10), with commands that have never been used displaying a yellow corner. Hovering over a *ToolTrip* will result in the commands that are part of the trip to display a badge of the colour of the trip (Figure 9D). Additionally, users can hover over individual command icons in the tooltip (Figure 9B) which will then highlight only that specific command in the interface. By default, Fusion 360 fades out the tooltip if the pointer is moved away from the command. We modified this behaviour to provide users with the opportunity to explore the expanded tooltip without having to keep the mouse cursor over the command. Specifically, we added a 500ms delay once the user moves the pointer away from the command. Only if the mouse cursor is not inside the tooltip at the end of this 500ms delay, then the tooltip fades out. Otherwise, it remains visible to let users interact with it.

Clicking on a *ToolTrip* in the tooltip will result in pinning that particular trip. A new view is opened at the right of the application showing the title of the trip and the list of icons for commands in the trip (Figure 9C). Users can hover over each icon in the trip to locate the corresponding command in the interface. While this view is pinned, commands in the interface that are part of the trip display a badge with a number indicating their position in the sequence. To stop following the trip, users can click the "Stop" button in Figure 9C.

[1] https://www.qt.io/

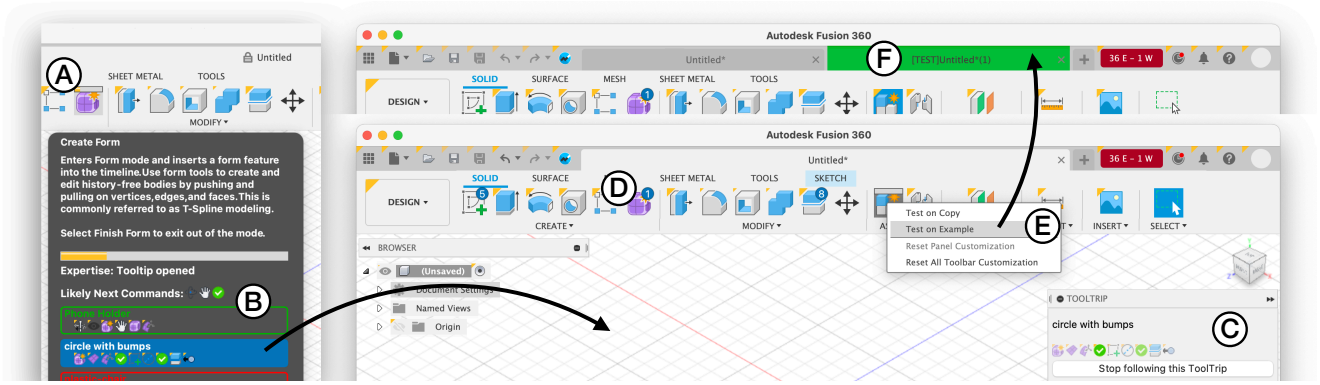

Figure 9: Our prototype implementation in Fusion 360 showing ToolTrack (A), ToolTrip (B, C, D) and ToolTaste (E, F).

Lastly, we modified the right-click contextual menu for commands by adding two options: "Test on copy" and "Test on example" (Figure 9E). Clicking these menu options will open a new tab with either an example-model that we created for that purpose beforehand, or a copy of the current model. Copies are made in-memory to ensure a smooth experience. This new tab is coloured green and titled "[TEST]" to make it stand out (Figure 9F). Users can then manipulate the model as they would with their main model. Alternatively, they can close the tab to discard the changes, or save the model if they are satisfied with the result.

7.2 Automatically Generating ToolTrips

A *ToolTrip* is essentially a sequence of commands associated with a title and a description. While *ToolTrips* could have been created manually using a set of representative tasks, given the large number of ways to arrange commands, offering personalized suggestions for all users would require creating thousands of *ToolTrips*. As an alternative approach, we propose to generate ToolTrips automatically from data collected from the community. Specifically, Fusion 360 users can record "Screencasts" [2] (based on Chronicle [33]) which are video recordings with meta-data about the specific commands being used. People often post these screencasts on forums to demonstrate how to use a tool or ask for help. As they contain users' workflows and other meta-data such as a title and a brief description, they are an ideal source for generating *ToolTrips*.

We collected 73,573 public screencasts generated by users of the Fusion 360 community. We excluded screencasts categorized as "Bug Reports" and "Troubleshooting", those with no title, and those published more than 6 months ago in order to avoid screencasts that use outdated workflows. We also filtered out very long (over 30 commands) and very short (less than 5 commands) screencasts. Finally, we obtained 1,936 screencasts, which we turned into *ToolTrips* by extracting their sequence of commands, title, and description.

7.3 Suggesting ToolTrips

One approach to allow users to explore and find *ToolTrips* would be to offer a search engine. This is in part what DiscoverySpace [11] proposed by prompting users for their task and offering suggestions based on this prompt. However, articulating search queries can be difficult for users, especially with new software applications [17], and would prevent serendipitous discovery.

Instead, we automatically suggest *ToolTrips* to the user and update suggestions based on the current context. Considering that *ToolTrips* are shown in a command's tooltip, we base our suggestion algorithms on the command under the mouse cursor and the user's recent command history. Our suggestion system works as follows: First, it filters out *ToolTrips* that do not contain the command under the mouse cursor. Then, it computes scores based on the number of commands in the *ToolTrip* that also appear in user's recent command history. Whenever a command in the *ToolTrip* also appears in the last five commands executed by the user, we add this command's inverse frequency to the *ToolTrip*'s score (frequencies are computed as the percentage of the number of times a command occurs throughout our database of *ToolTrips*). We found this scoring function to work decently well for our purpose; it does a fuzzy matching to find workflows matching the users' command history, attributing more weight to infrequent commands. Additionally, we made a variation of this scoring function to return workflows involving novel commands. We implemented this variation by adding the inverse frequency of all the commands that are part of the *ToolTrip* but were never explored before. In our final implementation, tooltips show three different *ToolTrips*: the top two returned by the first scoring function, and the top one returned by the second scoring function.

7.4 Example Scenarios

Throughout this section, we provide a walkthrough of how *ToolTrack*, *ToolTrip* and *ToolTaste* can be used to support users during trial-and-error sessions in Fusion 360.

7.4.1 Task 1: Model a Pen Holder. First, the user scans the interface by reviewing icons and labels. Three commands appear relevant: "Create Sketch", "Extrude", "Create Form" (Figure 10, Step 1, respectively Ⓐ, Ⓑ and Ⓒ). The user narrows down the options by reviewing *ToolTrips* to give them an idea of what commands can do. Specifically, "Create Form" has the *ToolTrip* "Phone Holder" which seems close to their goal of creating a pen holder (Figure 10, Step 2). The user then selects the *ToolTrip* and moves on to Fusion 360's "Form" tab (Figure 10, Step 3). Here, the *ToolTrip* highlights the "Box"

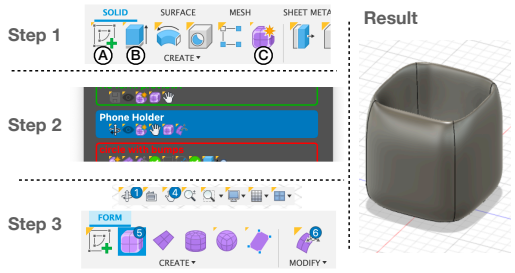

Figure 10: Creating a pen holder by following a ToolTrip titled "Phone Holder".

command 5, the "Modify" command 6, and several navigation commands such as "Pan" 4 and "Orbit" 1. After experimenting with these commands, the user quickly grasps how to navigate around the object, and manages to create a box using the "Box" command. Then, using the "Modify" command, the user is able to carve out a hole in the middle of the box (to hold the pens) by dragging the top faces of the box down (Figure 10, Result).

7.4.2 Task 2: Turning the Pen Holder into a Cup. The user now decides to add a handle so that they can use the pen holder as a cup. For this task, the previous "Create Form" command is not adapted as the user wants a sharp handle. They remember seeing the "Create Sketch" command that looked promising during their previous trial-and-error episode. Specifically, the command had a *ToolTrip* titled "Create Card Holder", which also seemed relevant. To not lose their progress, the user right-clicks the "Create Sketch" command and starts experimenting on a copy of their current document (Figure 11, Step 1). Once in sketch mode, the user starts exploring commands that appear relevant. Using *ToolTrack* as a guide, the user quickly realizes that while they explored all the relevant commands, they have not yet explored all the options for each command, judging by the progress bar when hovering commands (Figure 11, Step 2). They stop exploring different commands and instead focus on exploring options of the "2-Point Rectangle" command: by reviewing the yellow *ToolTrack* corners, the user sees that options such as "Look at" have not been tried yet. After experimenting with the different options, the user manages to create the sketch that they wanted next to the original pen holder. Once finished, the user sees that

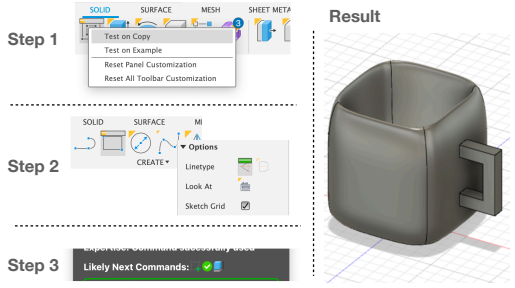

Figure 11: Exploring an alternative approach using ToolTaste to work on a copy, and ToolTrack to pragmatically explore relevant commands and options.

one of the likely next commands after using "2-Point Rectangle" is "Extrude" (Figure 11, Step 3). They experiment with this command and manage to create the handle for the cup (Figure 11, Result). However, they are not satisfied with this version and discard the changes, reverting to their prior version of the pen holder.

8 DISCUSSION

Despite years of progress in improving ease-of-use and facilitating learning, complex software applications still offer relatively poor support for key aspects of trial-and-error, instead mostly relying on explicit help approaches such as tutorials. Using our framework, we proposed three tools to increase complex software's support for trial-and-error behaviours: *ToolTrack*, *ToolTrip* and *ToolTaste*. With our conceptual model and design space, we hope to inspire researchers to design novel techniques for trial-and-error and conduct empirical studies, ultimately further expanding on our framework.

8.1 Limitations

Our framework of trial-and-error constitutes the main contribution of our work and we designed *ToolTrack*, *ToolTrip* and *ToolTaste* as a way to test the generative power of our framework. However, a limitation of our work is that we did not directly assess the effectiveness of these tools. A user study could help obtain more conclusive evidence supporting *ToolTrack*, *ToolTrip* and *ToolTaste* as effective solutions against the challenges of trial-and-error. However, while a lab-based usability study may be relatively straightforward to conduct, it may only provide limited additional insights and will have low ecological validity. A longitudinal study in which users could integrate our three techniques into their daily workflows with Fusion 360 would be the most useful for assessing the effectiveness and utility of our techniques in real-world situations, but would also be the most resource-intensive to run. Meanwhile, *ToolTrack*, *ToolTrip* and *ToolTaste* can serve as inspiration for how to use our framework in a generative way and how to design and implement trial-and-error techniques for complex software. Our techniques demonstrate how common mechanisms of WIMP interactions (e.g., tooltips, information-on-hover, menus, icons) in combination with community-sourced data such as video tutorials can be leveraged to develop novel cross-software techniques targeting trial-and-error.

8.2 Future Work

Our work opens up several avenues for future research. While *ToolTrack*, *ToolTrip* and *ToolTaste* improve support for trial-and-error, other aspects still remain poorly supported, as highlighted in our design space (Figure 5). For example, experimenting and recovering from modifications that target the user interface (such as closing a panel) are rarely supported, which might make users reluctant to customize the interface out of fear of not being able to recover from it. Mahmud et al. [24] also identified this gap and proposed to distinguish between "Undo" actions that affect the document from the actions that affect the interface. However, this remains to be tested in practice as it may also disrupt users' mental model of the undo stack.

An interesting direction for future research is to explore how tools for trial-and-error would work in and could support collaborative use. Software increasingly provides support for real-time

collaboration (e.g. Google Docs, Office 365, Figma). It remains an open question how techniques such as *ToolTrack*, *ToolTrip*, *ToolTaste* would work when groups of people are working together collaboratively. While we already leverage community workflows in *ToolTrips* (as do other approaches such as Patina [26]), it would be interesting to explore how the expertise and command history of collaborators can be leveraged in real-time to help teams achieve their goals and learn to use new functionality in the software. Could we identify who in the team has the needed expertise to accomplish a goal? Can we use a divide-and-conquer approach for collaborative trial-and-error? Would everyone have their own individual *ToolTracks* in addition to team *ToolTracks*?

People also often accomplish their work with workflows that extend beyond a single application. Workflows can span across multiple applications, even applications from different vendors. For example, an individual may create illustrations for a YouTube video in Adobe Illustrator, integrate the resulting graphics with the video in DaVinci Resolve, and reduce noise in the audio using Audacity. How might we support trial-and-error for such cross-application workflows that go beyond a single application? How can we highlight possible hand-off opportunities between different applications? Cross-application support could be considered an additional level of *Support* in our design space (Figure 5) and could be used to further generalize the conceptual model (Section 5.1.2).

Our three techniques are rooted in the common mechanics of desktop WIMP interfaces (as is most complex and feature-rich software). For example, we rely on *hover* interactions, which may not always be available on other platforms. Given the popularity of "professional" tablets for use on the go, it would be interesting to explore how our techniques would work on a tablet interface that relies mainly on pen and touch interaction. In these situations, it may be necessary to have a separate *mode* for trial-and-error that enables the interface augmentations that we proposed (like the progress bars in *ToolTrack*). Another interesting extension might be to use additional non-visual cues to reveal whether a command has been explored, such as audio cues (i.e., earcons [3]). Additionally, our techniques are currently geared towards commands that are activated with click interactions. How might we support trial-and-error for other interactions like drag operations (e.g., aligning objects, resizing objects with handles, drag-and-dropping a colour)? This also points to a larger research direction to consider trial-and-error for Post-WIMP interactions [42] and other modalities such as touch and mid-air gestures, voice interfaces, or Augmented, Mixed, and Virtual Reality.

Lastly, an evaluation of our techniques with designers and/or end-users would help assess their effectiveness at answering the challenges identified in our framework. As mentioned in Section 8.1, a long-term deployment of our modified version of Fusion 360 with a sizeable group of users would shed light on how people would use and integrate the three techniques in their everyday use of the application. This may also help us understand the right granularity of tracking a user's exploration with *ToolTrack*. Open questions that remain include what the appropriate "half-life" is of a command's past usage and when that usage should be discounted (e.g., if a command has not been used in the past three months). A deployment would also help to answer to what extent people prefer to have control over marking their own exploration of the software (i.e., having the ability to explicitly mark a tool as "unexplored", similar to marking an email as "unread" to capture that it should be revisited). Alternative design choices could also be evaluated, e.g., to determine the most effective recommendation algorithm for *ToolTrip* (should trips that include novel commands be preferred?). Similarly, interviews with software designers and participatory design sessions with end-users would help refine the designs of our techniques. While we designed the techniques to be software-independent before implementing them in Fusion 360, some software and users have unique challenges or use cases that our designs might not yet support (e.g., when a specific interactions like "hover" are already overloaded).

9 CONCLUSION

Early graphical interfaces developed in the 80s paved the way for learning without manuals through trial-and-error. Since then, software applications have grown in complexity, often making existing solutions for trial-and-error fall short. This paper is an attempt at improving our understanding of trial-and-error, and to identify the key aspects of trial-and-error that require more support with regard to complex and feature-rich software applications. We derived a framework based on observational studies of trial-and-error, identified challenges present in current approaches to support trial-and-error, and discussed the design and implementation of three techniques: *ToolTrack*, *ToolTrip* and *ToolTaste*. Through these techniques and our framework, we hope to provide a solution to help users trial-and-error in complex software applications as well as stimulate more research on systems that target trial-and-error.

REFERENCES

[1] Oscar D. Andrade, Nathaniel Bean, and David G. Novick. 2009. The Macro-Structure of Use of Help. In *Proceedings of the 27th ACM International Conference on Design of Communication - SIGDOC '09*. ACM Press, Bloomington, Indiana, USA, 143. https://doi.org/10.1145/1621995.1622022

[2] Autodesk. 2021. Autodesk Screencast. https://knowledge.autodesk.com/community/screencast Retrieved September 5, 2021.

[3] Meera M. Blattner, Denise A. Sumikawa, and Robert M. Greenberg. 1989. Earcons and Icons: Their Structure and Common Design Principles (Abstract Only). *SIGCHI Bull.* 21, 1 (Aug. 1989), 123–124. https://doi.org/10.1145/67880.1046599

[4] John M. Carroll and Caroline Carrithers. 1984. Training Wheels in a User Interface. *Commun. ACM* 27, 8 (Aug. 1984), 800–806. https://doi.org/10.1145/358198.358218

[5] John M. Carroll and Mary Beth Rosson. 1987. Paradox of the Active User. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. MIT Press, Cambridge, MA, USA, 80–111.

[6] Andy Cockburn, Carl Gutwin, Joey Scarr, and Sylvain Malacria. 2014. Supporting Novice to Expert Transitions in User Interfaces. *ACM Comput. Surv.* 47, 2, Article 31 (Nov. 2014), 36 pages. https://doi.org/10.1145/2659796

[7] Sjaak de Mul and Herre van Oostendorp. 1996. Learning User Interfaces by Exploration. *Acta Psychologica* 91, 3 (April 1996), 325–344. https://doi.org/10.1016/0001-6918(95)00060-7

[8] Stephen W. Draper and Stephen B. Barton. 1993. Learning by Exploration and Affordance Bugs. In *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems (CHI '93)*. Association for Computing Machinery, New York, NY, USA, 75–76. https://doi.org/10.1145/259964.260084

[9] Leah Findlater and Joanna McGrenere. 2004. A Comparison of Static, Adaptive, and Adaptable Menus. In *Proceedings of the 2004 Conference on Human Factors in Computing Systems - CHI '04*. ACM Press, Vienna, Austria, 89–96. https://doi.org/10.1145/985692.985704

[10] Leah Findlater, Karyn Moffatt, Joanna McGrenere, and Jessica Dawson. 2009. Ephemeral Adaptation: The Use of Gradual Onset to Improve Menu Selection Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Boston MA USA, 1655–1664. https://doi.org/10.1145/1518701.1518956

[11] C. Ailie Fraser, Mira Dontcheva, Holger Winnemöller, Sheryl Ehrlich, and Scott Klemmer. 2016. DiscoverySpace: Suggesting Actions in Complex Software. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*. ACM, Brisbane QLD Australia, 1221–1232. https://doi.org/10.1145/2901790.2901849

[12] C. Ailie Fraser, Julia M. Markel, N. James Basa, Mira Dontcheva, and Scott Klemmer. 2020. ReMap: Lowering the Barrier to Help-Seeking with Multimodal Search. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 979–986. https://doi.org/10.1145/3379337.3415592

[13] Tovi Grossman and George Fitzmaurice. 2010. ToolClips: An Investigation of Contextual Video Assistance for Functionality Understanding. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems - CHI '10*. ACM Press, Atlanta, Georgia, USA, 1515. https://doi.org/10.1145/1753326.1753552

[14] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. 2009. A Survey of Software Learnability: Metrics, Methodologies and Guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 649–658. https://doi.org/10.1145/1518701.1518803

[15] Pushkar N. Kaul. 1998. *Drug discovery: Past, present and future*. Birkhäuser Basel, Basel, 9–105. https://doi.org/10.1007/978-3-0348-8833-2_1

[16] Caitlin Kelleher and Randy Pausch. 2005. Stencils-Based Tutorials: Design and Evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '05*. ACM Press, Portland, Oregon, USA, 541. https://doi.org/10.1145/1054972.1055047

[17] Kimia Kiani, George Cui, Andrea Bunt, Joanna McGrenere, and Parmit K. Chilana. 2019. Beyond "One-Size-Fits-All": Understanding the Diversity in How Software Newcomers Discover and Make Use of Help Resources. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland UK, 1–14. https://doi.org/10.1145/3290605.3300570

[18] Benjamin Lafreniere, Andrea Bunt, and Michael Terry. 2014. Task-Centric Interfaces for Feature-Rich Software. In *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: The Future of Design* (Sydney, New South Wales, Australia) *(OzCHI '14)*. Association for Computing Machinery, New York, NY, USA, 49–58. https://doi.org/10.1145/2686612.2686620

[19] Benjamin Lafreniere, Parmit K. Chilana, Adam Fourney, and Michael A. Terry. 2015. *These Aren't the Commands You're Looking For*: Addressing False Feedforward in Feature-Rich Software. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, Charlotte NC USA, 619–628. https://doi.org/10.1145/2807442.2807482

[20] Frank Linton, Andy Charron, and Debbie Joy. 1998. *OWL: A Recommender System for Organization-Wide Learning*. Technical Report WS-98-08. The AAAI Press, Menlo Park, California.

[21] Frank Linton and Hans-Peter Schaefer. 2000. Recommender Systems for Learning: Building User and Expert Models through Long-Term Observation of Application Use. *User Modeling and User-Adapted Interaction* 10, 2 (June 2000), 181–208. https://doi.org/10.1023/A:1026521931194

[22] Aran Lunzer. 1998. Towards The Subjunctive Interface: General Support For Parameter Exploration By Overlaying Alternative Application States. *Late Breaking Hot Topics Proceedings of IEEE Visualization* 98 (1998), 45–48.

[23] Aran Lunzer and Kasper Hornbæk. 2008. Subjunctive Interfaces: Extending Applications to Support Parallel Setup, Viewing and Control of Alternative Scenarios. *ACM Trans. Comput.-Hum. Interact.* 14, 4, Article 17 (Jan. 2008), 44 pages. https://doi.org/10.1145/1314683.1314685

[24] Shareen Mahmud, Jessalyn Alvina, Parmit K. Chilana, Andrea Bunt, and Joanna McGrenere. 2020. Learning Through Exploration: How Children, Adults, and Older Adults Interact with a New Feature-Rich Application. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–14. https://doi.org/10.1145/3313831.3376414

[25] Justin Matejka, Tovi Grossman, and George Fitzmaurice. 2011. Ambient Help. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 2751–2760. https://doi.org/10.1145/1978942.1979349

[26] Justin Matejka, Tovi Grossman, and George Fitzmaurice. 2013. Patina: Dynamic Heatmaps for Visualizing Application Usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Paris France, 3227–3236. https://doi.org/10.1145/2470654.2466442

[27] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. 2009. CommunityCommands: Command Recommendations for Software Applications. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology - UIST '09*. ACM Press, Victoria, BC, Canada, 193. https://doi.org/10.1145/1622176.1622214

[28] Mathieu Nancel and Andy Cockburn. 2014. Causality: A Conceptual Model of Interaction History. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Toronto Ontario Canada, 1777–1786. https://doi.org/10.1145/2556288.2556990

[29] Don Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books, New York, New York, USA.

[30] David G. Novick, Oscar D. Andrade, and Nathaniel Bean. 2009. The Micro-Structure of Use of Help. In *Proceedings of the 27th ACM International Conference on Design of Communication - SIGDOC '09*. ACM Press, Bloomington, Indiana, USA, 97. https://doi.org/10.1145/1621995.1622014

[31] Peter Pirolli and Stuart Card. 1999. Information Foraging. *Psychological Review* 106 (10 1999), 643–675. https://doi.org/10.1037/0033-295X.106.4.643

[32] Peter G. Polson and Clayton H. Lewis. 1990. Theory-Based Design for Easily Learned Interfaces. *Human–Computer Interaction* 5, 2-3 (June 1990), 191–220. https://doi.org/10.1080/07370024.1990.9667154

[33] Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. 2011. Pause-and-Play: Automatically Linking Screencast Video Tutorials with Applications. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. Association for Computing Machinery, New York, NY, USA, 135–144. https://doi.org/10.1145/2047196.2047213

[34] Atul Prakash and Michael J. Knister. 1994. A Framework for Undoing Actions in Collaborative Systems. *ACM Trans. Comput.-Hum. Interact.* 1, 4 (Dec. 1994), 295–330. https://doi.org/10.1145/198425.198427

[35] Atul Prakash and Michael J Knister. 1994. A framework for undoing actions in collaborative systems. *ACM Transactions on Computer-Human Interaction (TOCHI)* 1, 4 (1994), 295–330.

[36] John Rieman. 1996. A Field Study of Exploratory Learning Strategies. *ACM Transactions on Computer-Human Interaction* 3, 3 (Sept. 1996), 189–218. https://doi.org/10.1145/234526.234527

[37] Andrew Sears and Ben Shneiderman. 1994. Split Menus: Effectively Using Selection Frequency to Organize Menus. *ACM Transactions on Computer-Human Interaction* 1, 1 (March 1994), 27–51. https://doi.org/10.1145/174630.174632

[38] Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (Aug. 1983), 57–69. https://doi.org/10.1109/MC.1983.1654471

[39] Herbert A. Simon and Peter A. Simon. 1962. Trial and error search in solving difficult problems: Evidence from the game of chess. *Behavioral Science* 7, 4 (1962), 425–429. https://doi.org/10.1002/bs.3830070402 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/bs.3830070402

[40] Michael Terry and Elizabeth D. Mynatt. 2002. Side Views: Persistent, on-Demand Previews for Open-Ended Tasks. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology* (Paris, France) *(UIST '02)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/571985.571996

[41] Michael Terry, Elizabeth D. Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. 2004. Variation in Element and Action: Supporting Simultaneous Development of Alternative Solutions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 711–718. https://doi.org/10.1145/985692.985782

[42] Andries van Dam. 1997. Post-WIMP User Interfaces. *Commun. ACM* 40, 2 (Feb. 1997), 63–67. https://doi.org/10.1145/253671.253708

[43] Wesley Willett, Jeffrey Heer, and Maneesh Agrawala. 2007. Scented Widgets: Improving Navigation Cues with Embedded Visualizations. *IEEE transactions on visualization and computer graphics* 13 (12 2007), 1129–36. https://doi.org/10.1109/TVCG.2007.70589