Cognitive Style Heuristics

The **Cognitive Style Heuristics (CSH)** are nine principles of **interaction design** used to improve software usability. They are framed around how different people use software in different ways. The CSH were created by a research team headed by Margaret Burnett, one of the world's leading experts in usability research and **inclusive software design**.

The CSH were created with new users in mind: People who have never seen, interacted with, or received previous direction on the software. They can also improve usability for more experienced users, who have figured out how to use the software to complete their tasks, but who may be unhappy with what the software requires of them. For example, there's software I regularly use for work that requires me to run (and wait for) a data export each time I need a row's unique identifier. I've learned the process, but couldn't the developers have displayed the unique IDs on-screen?!

Cognitive Style Heuristics (CSH): Nine principles of interaction design for finding and fixing usability bugs in software. They are based around different cognitive styles different people use when they problem-solve in software.

interaction design: An approach to technology design that involves helping users understand what's happening with the technology, what just happened, and what they can do (Norman 2013).

6.1 Cognitive Style Facets

inclusive software design: A type of software user interface design with the goal of increasing usability for traditionally under-served user populations while also increasing usability for mainstream users.

cognitive style facets (CSFs): Five aspects of users that affect how they solve problems in software: Motivations, information processing style, computer self-efficacy, attitude toward risk, learning style

motivation CSF: Why someone is using the software (task completion vs. interest)

information processing style CSF: How a person looks through or absorbs information in software (comprehensively vs. selectively)

motivation CSF: Why someone is using the software (task completion vs. interest)

computer self-efficacy CSF: A person's confidence in their ability to use computers or software (low vs. high)

At the core of the Cognitive Style Heuristics are the **cognitive style facets**, five aspects of human cognition that affect how users problem-solve in software:

1. **Motivations** for using software (task completion vs. interest). A person who is feeling task-motivated may choose to use software because they have something specific they need to accomplish, and may focus on that task immediately when they open the software and the whole time they're using it. A person who is feeling interested in the software itself may seek out software that has exciting new features they've never seen before and may spend a lot of time exploring those features.

2. **Information processing style** (comprehensive vs. selective). A person processing comprehensively may want to understand details, implications, or to get a sense of overall structure before taking action in software. A person processing selectively may taking action as soon as they detect what seems like the beginning of a promising path.

3. **Computer self-efficacy** (low vs. high). A person who is feeling low computer self-efficacy may think it's their fault when they make a mistake or encounter a problem in the software. A person feeling high computer self-efficacy may feel the software is poorly made if they make a mistake or encounter a problem in the software.

4. **Attitude toward risk** (risk-tolerant vs. risk-averse). A person who is feeling risk-averse may avoid taking actions that have unknown consequences or seem dangerous or irreversible. A person who is feeling risk-tolerant may take actions even if they know those actions could lead to bad consequences.

5. **Learning style** (tinkering vs. by process). When someone is tinkering, they might click someone just because it's clickable then learn from what happens. When someone is learning by process, they might seek a logical first step, and want to proceed smoothly from start to finish.

As you probably noticed, each of the cognitive style facets has two polar **cognitive style facet values** (or **cognitive**

styles). Each pair of facet values creates a spectrum. When each of us uses software, the way we feel and behave corresponds to somewhere on each of those spectra. Our individual facet values may be similar each time we use software, but they can also vary by context and change over time. For example, many people feel cognitively impatient when reading paragraphs of text ("text walls") on websites and might process them more selectively, whereas they might want to catch every word of a new novel by their favorite author (comprehensive processing).

6.2 Cognitive Style Personas

The CSH are stated from the perspective of improving usability for the three **cognitive style personas**: Abi, Pat, and Tim. A **persona** is fictional person that is created to represent a group of users within a target audience. Personas are used to help marketing teams keep important subsets of their target audience in mind, and in software UI design for the same reason. Personas are typically documents that include a photo, name, age, gender, other background information, and information about how the made-up person interacts with product or software.

The cognitive style personas have a similar purpose but are distinct from traditional personas in multiple ways:

- There are three and only three (Abi, Pat, and Tim).
- Abi, Pat, and Tim each have a different set of cognitive styles. The cognitive styles are fixed.
- Abi, Pat, and Tim are each multi-personas: They each have multiple photos of different people who appear to be of different ages, races, genders, etc.
- Abi, Pat, and Tim were specifically created for evaluating *software* (not marketing).
- Abi, Pat, and Tim represent different positions on the cognitive style facet spectrum: Abi and Tim are on the ends and Pat is in the middle.

The idea of the cognitive style personas is that creating software that works well for Abi and Tim (the two ends of the cognitive style facet value spectrum) will result in software that's better for them and everyone in between.

attitude toward risk CSF: How willing a person is to take chances in software (risk-tolerant vs. risk-averse)

. . . . . .

learning style CSF: How a person prefers to move through software (tinkering vs. by process)

. . . . . .

cognitive style facet value (A.K.A., cognitive style): A position on the spectrum of a cognitive style facet

. . . . . .

persona: A fictional character that represents a subset of users in a target audience. Personas are used in marketing and UI design to help with focusing on particular groups of users and customers (Pruitt 2010; Martin 2012).

. . . . . .

cognitive style personas: Three specialized personas (Abi, Pat, and Tim) used for making software UI designs more usable to people with different cognitive styles.

. . . . . .

Intentional?

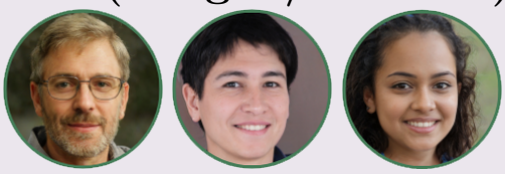
PAT

ABI

TIM

Abi, Pat, and Tim

Abi (Abigail/Abishek)



Motivation: Uses technology to accomplish their tasks.
Computer self-efficacy: Lower self-confidence than their peers about doing unfamiliar computing tasks. Blames themselves for problems.
Attitude toward risk: Risk-averse about using unfamiliar technologies that might require a lot of time
Information processing style: Comprehensive
Learning style: Process-orientated learning

Pat (Patricia/Patrick)

Motivation: Learns new technologies when they need to
Computer self-efficacy: Medium confidence doing unfamiliar computing tasks. If a problem can't be fixed, they will keep trying
Attitude toward risk: Risk-averse and doesn't want to expend time when they might not receive benefits
Information processing style: Comprehensive
Learning style: Likes to explore and purposefully tinker

Tim (Timara/Timothy)

Motivation: Likes learning all the available functionality on all their devices
Computer self-efficacy: High confidence in technical abilities. If a problem can't be fixed, blame goes to software vendor.
Attitude toward risk: Doesn't mind taking risk using features of technology
Information processing style: Selective
Learning style: Likes tinkering and exploring

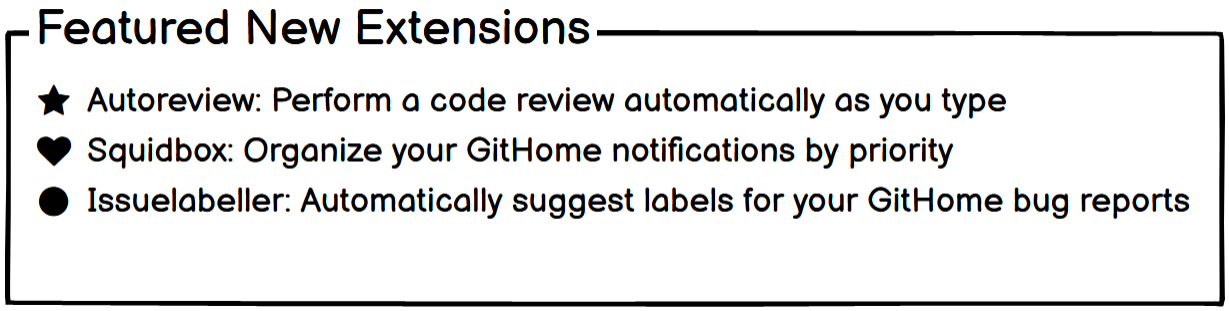
6.3 The Heuristics

Note: The designs shown below were modelled after examples found in published software.

Heuristic #1 (of 9): Explain what *new* features do, and why they are useful

- Allow Abi and Pat to quickly assess new features so they can choose whether to start using them. Allow Tim to quickly assess what a feature is for so they can move on to finding out what the rest of the software's features do.
- Abi and Pat use software only as needed for their task. They prefer familiar features to keep focused on the task and may be wary of new features.
- Tim likes using software to learn what new features can help them accomplish.

Example 1: Each featured extension has a brief description that says what the extension does and why somebody would use it.



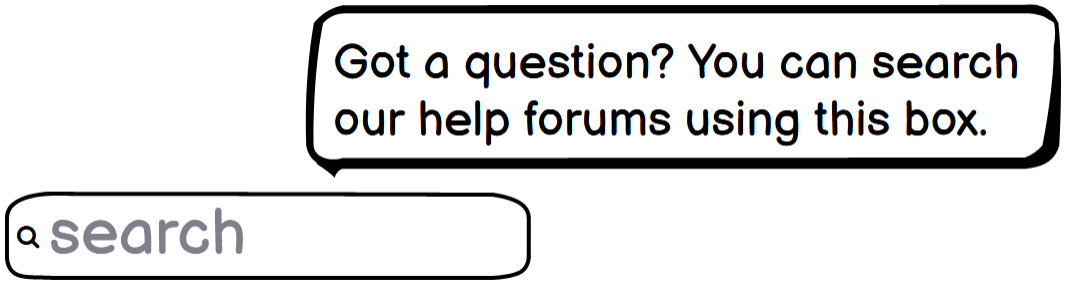
Example 2: This Keybase announcement briefly describes a new feature and how to use it.

Heuristic #2 (of 9): Explain what *existing* features do, and why they are useful

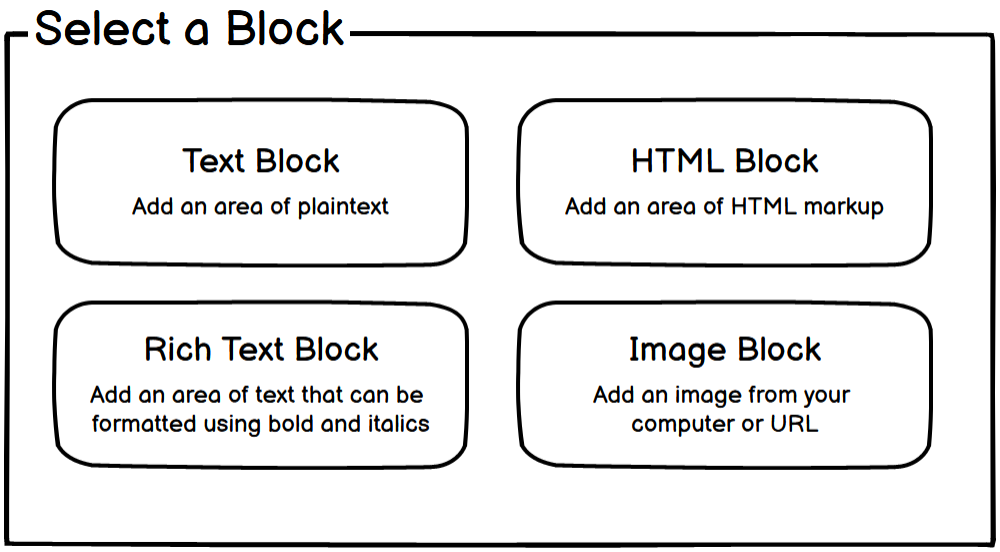
- Allow Abi to determine whether existing features are familiar to them. Allow Tim and Pat to more efficiently determine which features are known and which are new and unique.
- Abi is risk-averse and so may avoid using features that have an unknown time cost and an unknown benefit.

- Pat is also risk-averse, but might tinker with features to determine whether they are relevant to accomplishing their task.
- Tim is risk-tolerant so may use features without knowing their cost or even what they do. Tim is also motivated to investigate new, cutting-edge features.

Example 1: The tooltip explains what the search is for and why someone might use it.



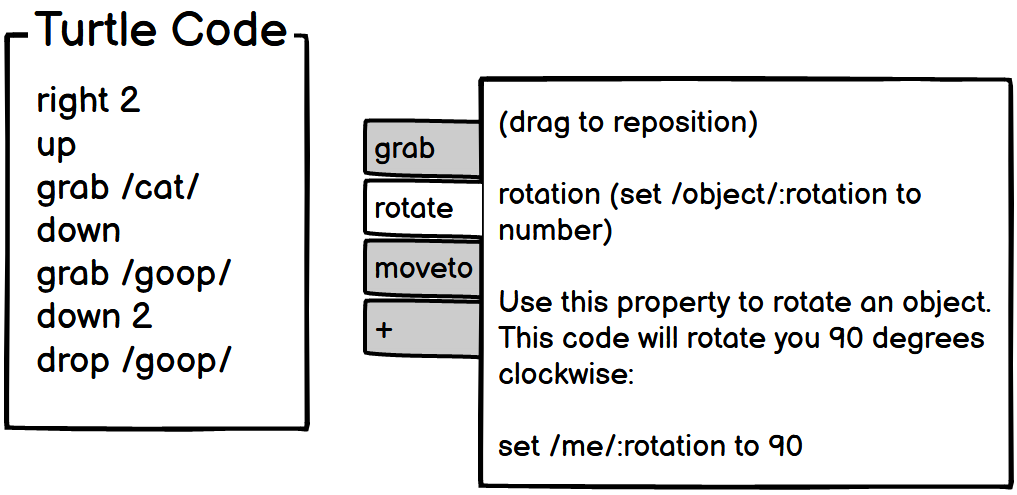
Example 2: Each tile explains a feature and the benefit of using the feature.



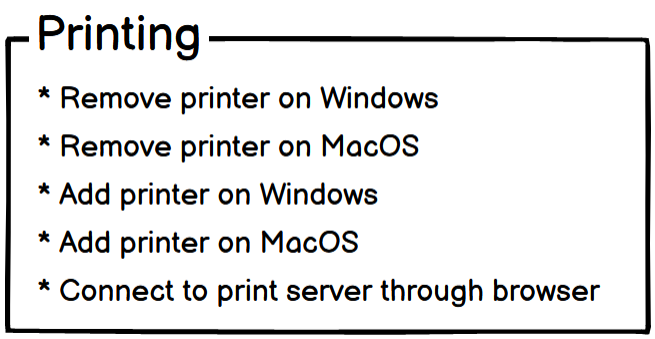
Heuristic #3 (of 9): Let people gather as much information as they want, and no more than they want

- Allow Abi and Pat to find the information they want, but don't force them to spend excessive time or effort gathering that information. Allow Tim to find correct information immediately, so that they don't go down the wrong path or get distracted from their task.
- Abi and Pat gather and read relevant information comprehensively before acting.
- Tim likes to delve into the first option and pursue it, backtracking if need be.

Example 1: Users can choose to view code documentation while still viewing their code.



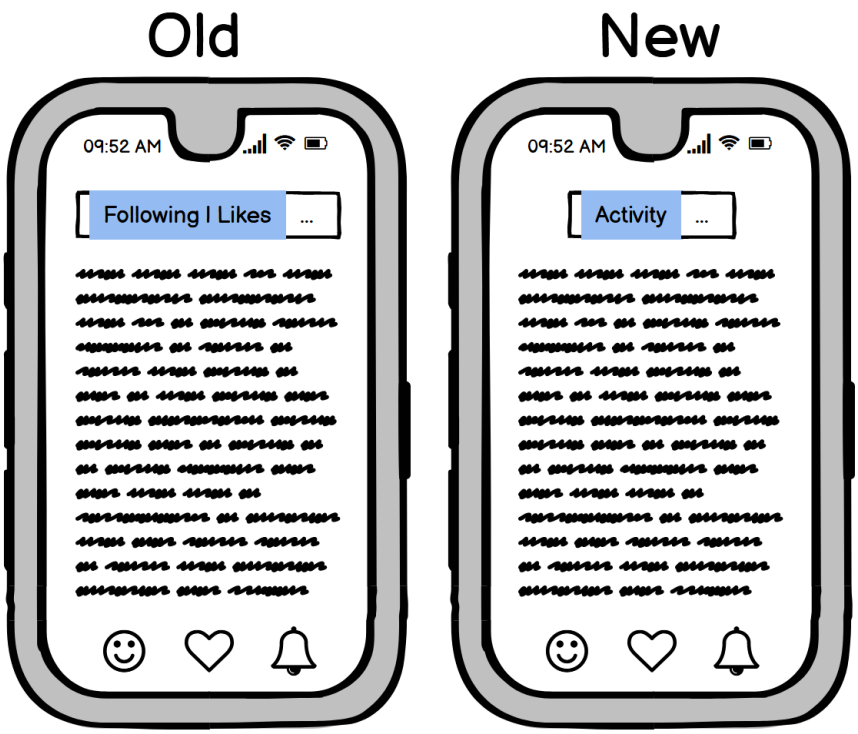
Example 2: Users can quickly see the contents of the webpage and jump to the section they're interested in.



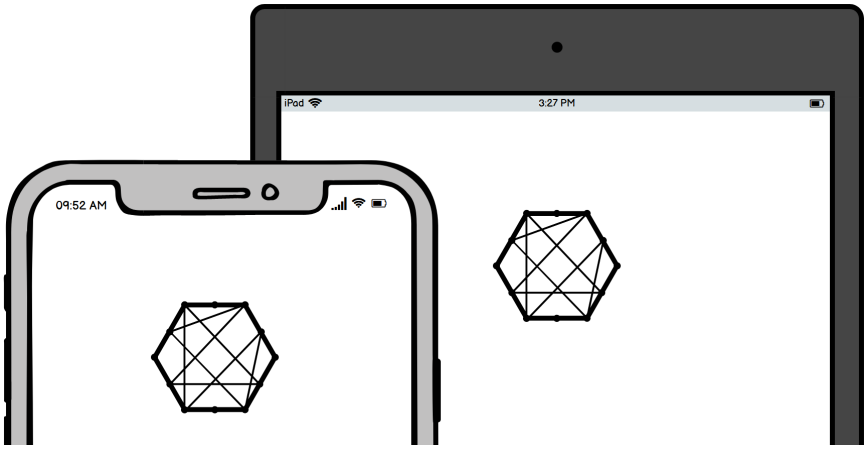
Heuristic #4 (of 9): Keep familiar features available

- To encourage Abi, Pat, and Tim to continue using the software, allow them to interact with it in ways they expect.
- Abi has low computer self-efficacy, so often takes the blame if a problem arises in the software, such as features having changed.
- Pat has medium self-efficacy with technology, and will keep on trying for a while if a problem arises, such as features having changed.
- Tim has high computer self-efficacy, so often blames the software if a problem arises, such as features having changed.

Example 1: Although the "following" page is gone, the new update looks similar to the previous version so that users are still familiar with the app.



Example 2: The smartphone and tablet versions of this app offer the same features which makes switching between the two easy.

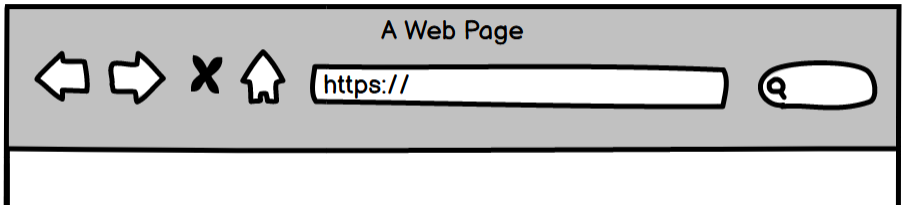


Heuristic #5 (of 9): Make undo/redo and backtracking available

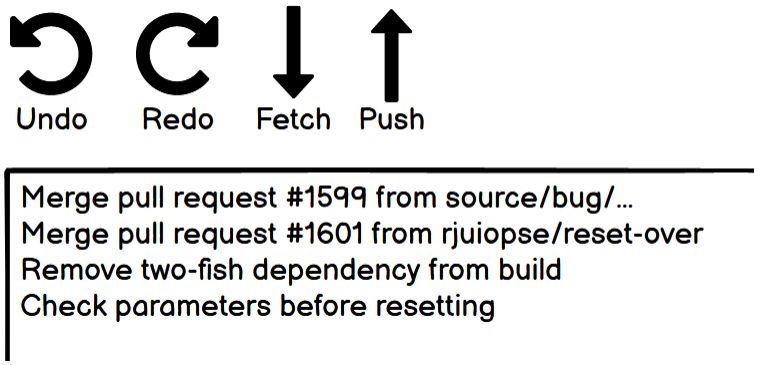
- Provide undo/redo and backtracking to allow Abi and Pat to feel comfortable proceeding, and to allow Tim to recover from mistakes.
- Abi and Pat are risk-averse, so they prefer not to take actions in software that might not be easy to reverse.
- Tim is risk-tolerant, so is willing to take actions in software that might be incorrect and need to be reversed.

Example 1: Browser back/forward buttons allow users to backtrack through

their browsing history.



Example 2: An undo button allow users to make and recover from mistakes. Also, version control control systems allow users to revert to any previously-committed code state.

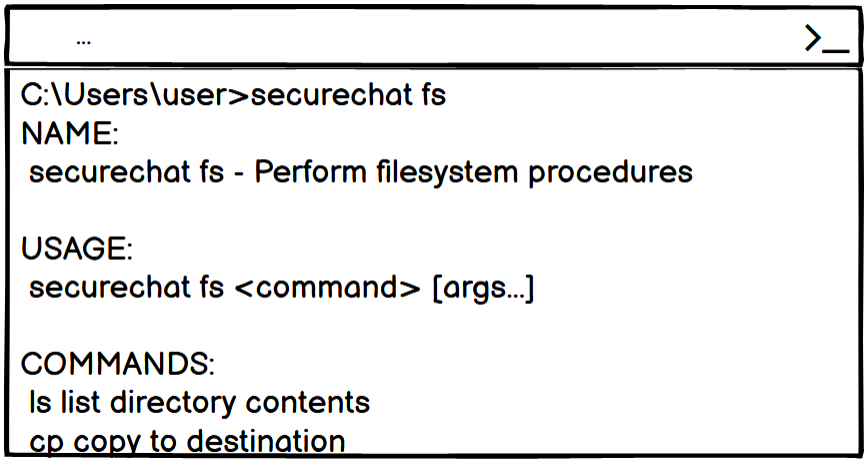


Heuristic #6 (of 9): Provide ways to try out different approaches

- Allow Abi to try a different approach when they feel unable to proceed with the current one. Allow Tim and Pat to try multiple ways of solving a problem.
- Abi has low computer self-efficacy, so often takes the blame if a problem arises in the software. This can discourage Abi from persevering in the software.
- Pat has medium self-efficacy with technology, and will keep on mindfully tinkering for a while if a problem arises, to find the solution.
- Tim has high computer self-efficacy, so often blames the software if a problem arises but is willing to try numerous ways of addressing the problem.

Example 1: If users don't find what they need on the "Choose a Question" drop-down menu, they can try the chat.

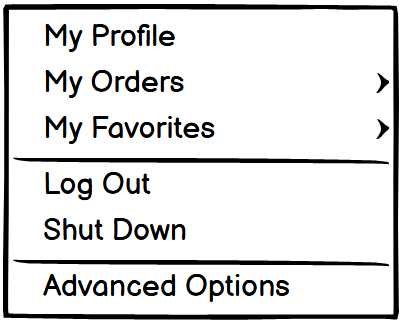
Example 2: If users encounter a problem using the SecureChat UI, they can attempt the same operations using the command line interface.



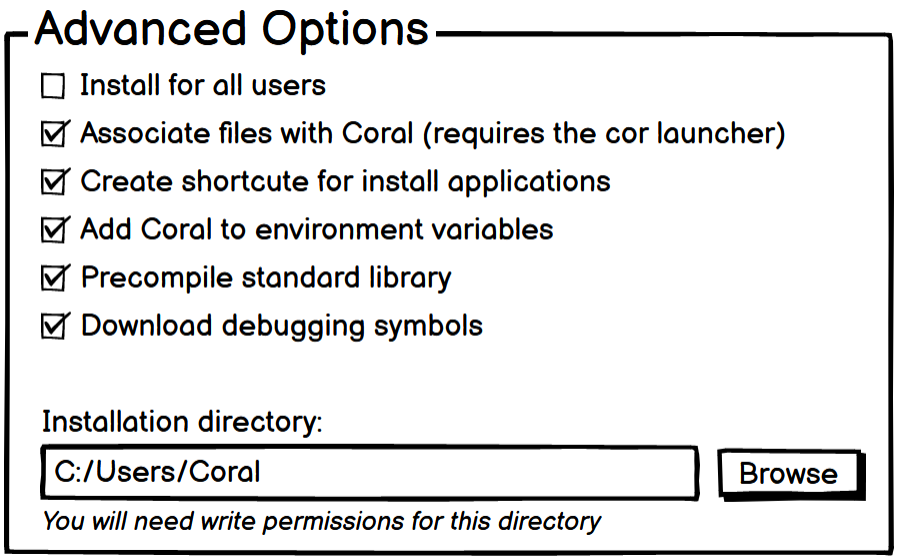
Heuristic #7 (of 9): Communicate the amount of effort that will be required to use a feature

- Allow Abi and Pat to decide whether or not a feature will require too much effort to use. Allow Tim to understand that a feature may take extra effort, and thus more time, to help them stay on track with their task.
- Abi and Pat are risk-averse, so they may want to avoid features with high effort costs.
- Tim is risk-tolerant, so may begin using features that require extra effort and time, and that are unrelated to the task at hand.

Example 1: Placing "Advanced Options" at the bottom of the menu indicates to the user that "advanced" features may take more effort.



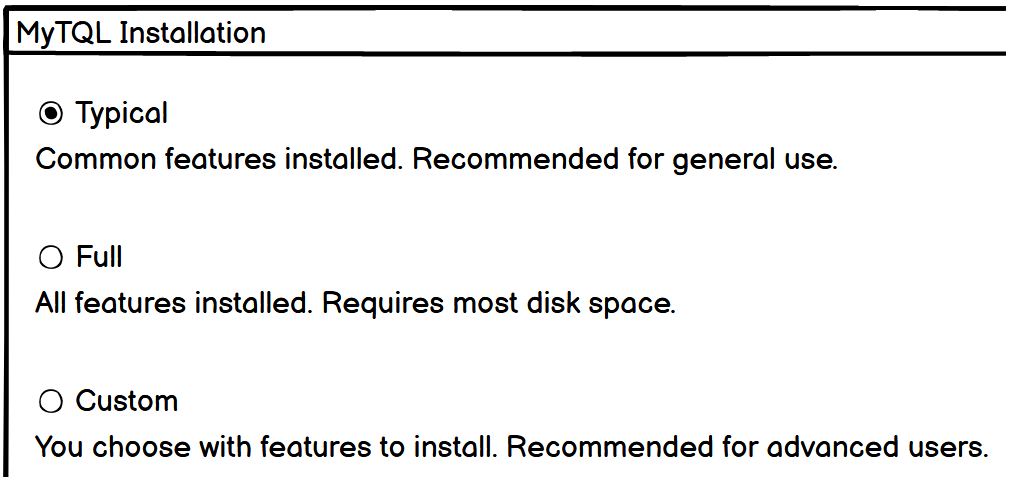
Example 2: The dialog indicates that "cor launcher" will be needed to "associate files with Coral" and that the user will need write permissions for the installation folder.



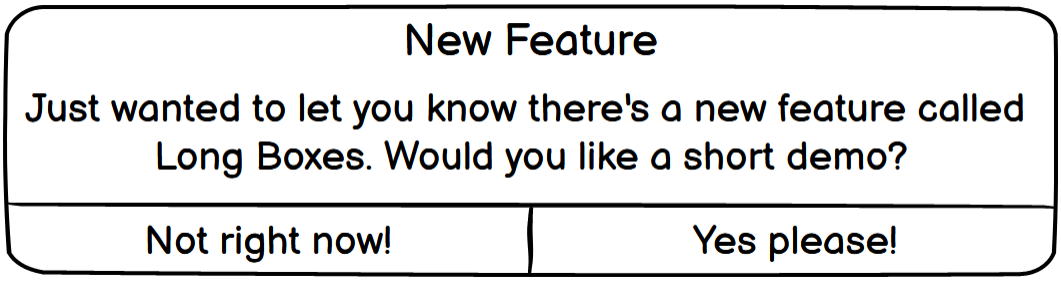
Heuristic #8 (of 9): Provide a path through the task

- Provide Abi a way to go through tasks using a clear process. Provide Tim and Pat a way to bypass step-by-step processes and tutorials if those are not required for learning the software.
- Abi is a process-oriented learner, so prefers to proceed through tasks step-by-step.
- Tim and Pat learn by tinkering, and therefore prefer not to be constrained by rigid, pre-determined processes.

Example 1: Users can choose their entry point, and each path is explained.



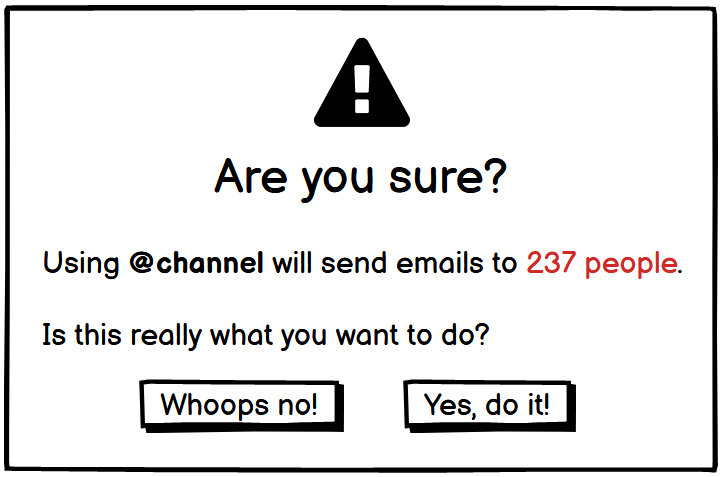
Example 2: Users get to choose either the path of learning more about the new feature or going back to what they were doing.



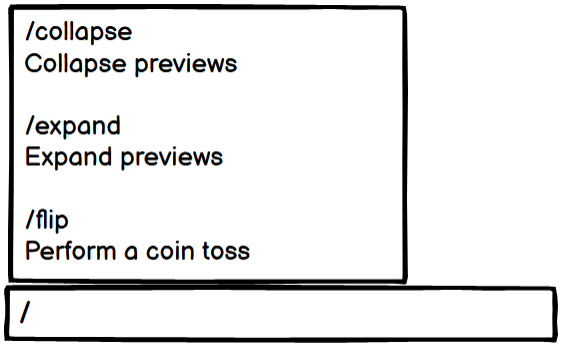
Heuristic #9 (of 9): Encourage mindful tinkering

- Encourage Abi to tinker in ways that lead to them discovering task-relevant functionality. Encourage Tim not to over-tinker (e.g., by adding an extra click), so that they make fewer mistakes, have time to absorb important information, and stay on-task.
- Abi is a process-oriented learner so usually prefers not to tinker. Because of this, they might miss useful or important parts of the software.
- Pat learns by trying out new features but does so mindfully, reflecting on each step.
- Tim learns by tinkering, but sometimes tinkers addictively and gets distracted from their task.

Example 1: This design encourages users to tinker mindfully by showing they will notify them before impactful actions are executed, like emailing 237 people.



Example 2: Keybase encourages users to try out new "slash" commands by showing all the commands when a user types "/", and explaining what each does and how to use it.



6.4 Background

heuristic evaluation: A usability inspection method where evaluators independently check that a design reflects a set of heuristics, then compare results (Nielsen and Molich 1990).

The Cognitive Style Heuristics are meant to be used in a **heuristic evaluation**, a process where software designers or evaluators go through heuristics one-by-one like a checklist, deciding whether the design does or does not reflect the heuristic. The evaluation is done independently by two or more people, who then compare findings.

The Cognitive Style Heuristics are derived from the GenderMag Heuristics (Burnett, Sarma, et al. 2018) and the **GenderMag Method** (Burnett, Stumpf, et al. 2016). The GenderMag Method a process for finding and fixing gender-inclusivity bugs in software. Instead of heuristic evaluation, it uses a **cognitive walkthrough**. It uses the same personas: Abi, Pat, and Tim. However, in addition to their cognitive styles, each GenderMag persona has additional sections, such as one with customizable background information.

The cognitive style personas are simplified versions of the GenderMag personas. You can find the the GenderMag personas, and a full description of their research origins, at GenderMag.org

GenderMag Method: A method for finding and fixing gender-inclusivity bugs in software that uses a specialized cognitive walkthrough and the customizable Abi, Pat, and Tim personas (Burnett, Stumpf, et al. 2016)

cognitive walkthrough: A usability inspection method that involves stepping through a user interface as a user, stopping to ask specific questions about the user's experience (Nielsen and Mack 1994).

What do cognitive styles have to do with gender? Software tends to be biased against the cognitive styles often favored by women. Designing with cognitive styles in mind can make software less gender-biased (Vorvoreanu et al. 2019).

In addition, "designing software so that it works for diverse populations matters to software companies' profitability, to equity in the workplace and at home, and to anyone in a situation that changes the way they think, such as when under deadline pressure."(Mendez et al. 2019)

6.5 Summary

The Cognitive Style Heuristics are a set a nine software usability heuristics for evaluating and improving the usability of UIs across users with different cognitive styles.

References

Margaret Burnett, Simone Stumpf, et al. (Oct. 2016). "GenderMag: A Method for Evaluating Software's Gender Inclusiveness". In: *Interacting with Computers* 28.6, pp. 760–787. ISSN: 0953-5438. DOI: 10.1093/iwc/iwv046. eprint: https://academic.oup.com/iwc/article-pdf/28/6/760/7919992/iwv046.pdf. URL: https://doi.org/10.1093/iwc/iwv046

GenderMag.org (n.d.). http://gendermag.org. Accessed: 2020-12-27

Margaret Burnett, Anita Sarma, et al. (July 2018). *The GenderMag Heuristics (Beta Version)*. https://gendermag.org/flyers_handouts.php

Bella Martin (2012). *Universal methods of design : 100 ways to research complex problems, develop innovative ideas, and design effective solutions*. Digital ed. Beverly, MA: Rockport Publishers. ISBN: 9781610581998

Christopher Mendez et al. (2019). "From GenderMag to InclusiveMag: An Inclusive Design Meta-Method". eng. In:

Jakob Nielsen and Rolf Molich (1990). "Heuristic Evaluation of User Interfaces". In: *IN: PROCEEDINGS OF THE CHI´90 CONFERENCE, SEATTLE*. S, pp. 249–256

Jakob Nielsen and Robert L. Mack (1994). *Usability inspection methods*. New York

Don Norman (2013). *The Design of Everyday Things: Revised and Expanded Edition*. eng. Rev. and expanded ed. Boulder: Basic Books. ISBN: 9780465050659

John Pruitt (2010). *The essential persona lifecycle : your guide to building and using personas*. San Francisco, Calif. : Oxford: Morgan Kaufmann ; Elsevier Science [distributor]. ISBN: 9780123814180

Mihaela Vorvoreanu et al. (2019). "From Gender Biases to Gender-Inclusive Design: An Empirical Investigation". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. Glasgow, Scotland Uk: Association for Computing Machinery, pp. 1–14. ISBN: 9781450359702. DOI: 10.1145/3290605.3300283. URL: https://doi.org/10.1145/3290605.3300283