Evaluation
Plan for today

A structured conversation about the relationship between today’s reading and our role as PL+HCI researchers
This paper played a big role in the HCI community in broadening the classes of evaluations considered acceptable, including no-evaluation papers.

What’s this to do with us?

- A lot of parallels to evaluating PLs. (In your head, replace “UI system” or “UI toolkit” with “PL” and see how many observations still hold.)
- Framework for how to think about meaningfully evaluating complex design contributions

Thank Amy Ko for these insights, and check out her work for more of the same!
Value added by UI systems architecture (...and PLs!)

• Reduce development viscosity
• Least resistance to good solutions
• Lower skill barriers
• Power in common infrastructure
• Enabling scale
Evaluation Errors

- The usability trap
- The fatal flaw fallacy
- Legacy code
Usability Trap

Common measures

- Time to complete standard task
- Time to reach proficiency
- Number of errors

Sound familiar?
Another take on the usability trap, well worth a read

- Usability eval as weak science
  - Do we end up picking problems and solutions that are amenable to these evals rather than picking research question, then choosing eval that fits?
  - We often do it as existence proof rather than testing risky hypothesis.
- Using usability eval too early
  - Quashing cool ideas by testing for usability before they’re usable, even if they have promise
  - Consider too few ideas; many parallel ideas standard in other design and engineering fields
- Innovation, Cultural Adoption
  - Usable vs. useful
  - Discovery: find facts about the world
  - Innovation, invention: create new and useful things
  - Many very useful inventions (e.g., cars) started out pretty unusable
  - Even our best inventors often don’t anticipate how culture will use the inventions
Usability Trap

Common assumptions
• Walk up and use, minimal training
  • Using doesn’t require expertise, or if it requires specific expertise many people already have that expertise
• Standardized task assumption
  • If we’re going to compare across two systems…
• Scale of the problem
  • Task usually needs to be completable in 1-2 hours
The fatal flaw fallacy

Say every time someone proposes a new PL or new abstraction, we try to find a program that can’t be expressed with it. Is that a good way to evaluate?
Legacy code

Is it bad to propose new languages when people are already so experienced with existing ones? When they have so many libraries available? So much code already written?
What else can we use to evaluate if PLs, abstractions, programming systems, programming tools contribute something valuable?

If we won’t eval usability, covering everything, and if we allow we don’t have to be backwards compatible with all legacy code?
For the next few slides, we’re going to take the reading’s contribution types one at a time.

In your breakout groups, please brainstorm ways to demonstrate these claims for PL/Programming Systems contributions.
I recommend having the reading open in front of you if possible, for inspiration. But I also recommend brainstorming on your own before you refer back to it!

If you struggle to come up with ideas, try making it more concrete. How would you assess this contribution for work in the domain of your final project? The final projects you critiqued last week?
Before all other claims a system, toolkit or interactive technique must demonstrate importance. Tools are invariably associated with expertise gained over time. People will not discard a familiar tool and its associated expertise for a 1% improvement. In most cases at least a 100% improvement is required for someone to change tools. Without establishing the importance of the problem and its proposed solution, nothing else matters.
Problem not previously solved

This is one of the more compelling claims for a tool. This claim says that there is a STU context that has no current solution. It is a powerful claim to demonstrate that T can be performed effectively with a new tool. Usability testing is irrelevant when comparing what can be done against what cannot.
Generality

The new solution claim is much stronger if there are several populations $U_i$ that each have tasks $T_i$ that do not have effective solutions with existing technology. If the new tool can solve all of $T_i$ then a claim for a general tool is quite strong. The generality of the new solution claim is strengthened as the populations $U_i$ are increasingly diverse from each other.
Reduce solution viscosity

One of the important attributes of good tools is that they foster good design by reducing the effort required to iterate on many possible solutions. The more cumbersome the tool, the greater the viscosity in the design process with fewer and less diverse alternatives being explored. There are at least three ways in which a tool can reduce solution viscosity: flexibility, expressive leverage and expressive match.
Empowering new design participants

The previous set of claims focused on the speed or ease with which a user interface could be designed. Tools can also make a contribution by introducing new populations to the UI design process. Frequently this is done by dealing with expressive leverage and expressive match issues, but the claims are different. The “new design participants” claim is that there is some population U who would benefit by being more directly involved with the UI design process. It has long been claimed that empowering artists will lead to better visual designs. Participatory design advocates the involvement of end-users in the design process.
Many tools demonstrate their effectiveness by supporting combinations of more basic building blocks. There are two basic variations of this claim. The first is an inductive claim that an infinite set of solutions can be built from primitives and their combinations. The second is the N to 1 reduction. Both of these approaches are based on clearly defining mechanisms for combining pieces of design to create a more powerful whole.
Can it scale up?

An important question that must be asked of every new UI system is whether it can scale up to large problems. This was the fundamental drawback of state machines for describing user interface dialogs. For simple examples like dragging a rubber-band line, the state machine dialog was clear and direct. However, for any reasonable application the representation acquired hundreds of states interconnecting in hundreds of ways that were impossible to visualize, present on a screen, or debug. Constraint systems have similar problems. They nicely model small local relationships yet produce serious debugging challenges when hundreds of constraints are all being evaluated simultaneously. Any new UI system must either show that it can scale up to the size of realistic problems or that such scaling is irrelevant because there is an important class of smaller problems that the new system addresses. To evaluate this criteria one must try the system on a reasonably large problem and show that the advantages of the new model still hold.
What do we get to claim?

• The fact that there are other ways to demonstrate value of PL/Programming Systems contribution doesn’t mean we get to make unsupported usability claims
  • Demonstrating one of these contributions doesn’t mean the tool is usable or that we get to make usability claims without usability eval
• Don’t get to make unsupported claims about these alternative contributions either!
  • But we do get to think creatively about how we evaluate them
So why’d we do this?

- Usability isn’t the only thing we can evaluate.
- Sometimes it’s not practical to evaluate it for PLs.
- …but we have alternatives available! We don’t have to just give up on human factors evaluations.
- The range of options means we have to be thoughtful about our goals, what we want to claim, what we evaluate.
Takeaways

- Highly encourage you before designing an evaluation to decide which of these dimensions (or others) about which you want to make claims
  - Sit down with the list, write out the specific claim
  - Then design the eval