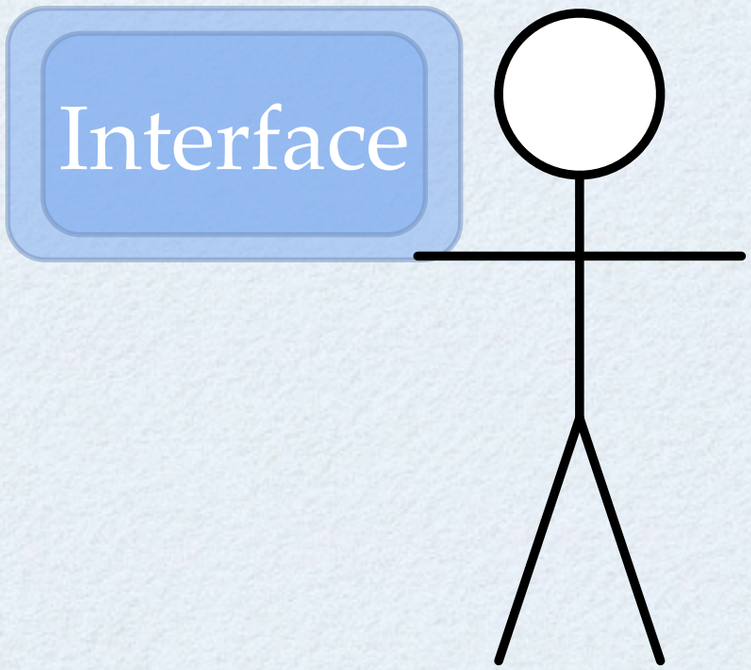
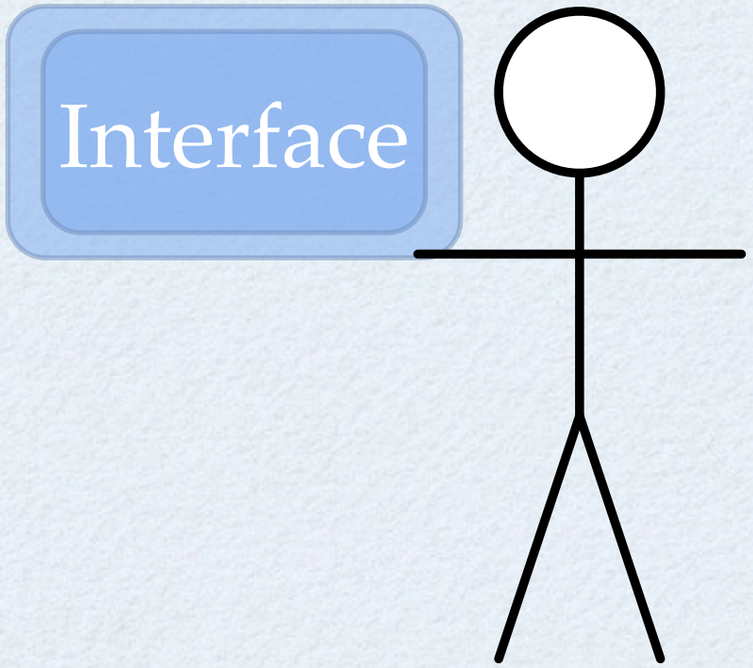
# Heuristic Evaluation of Programming Language Features: Two Parallel Programming Case Studies

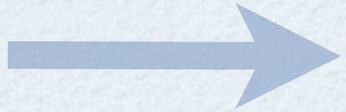
Caitlin Sadowski
Sri Kurniawan
University of California at Santa Cruz

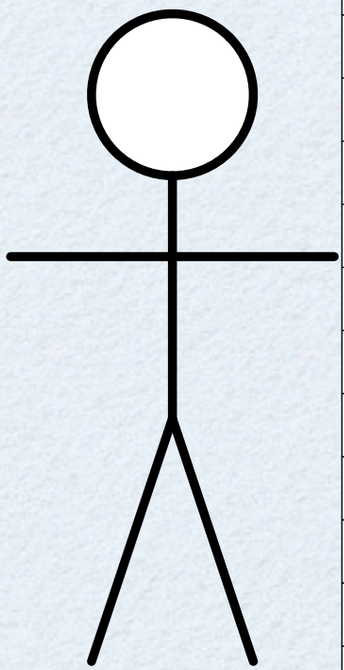
# Can heuristic evaluation help programming language researchers find problems?

# Yes!

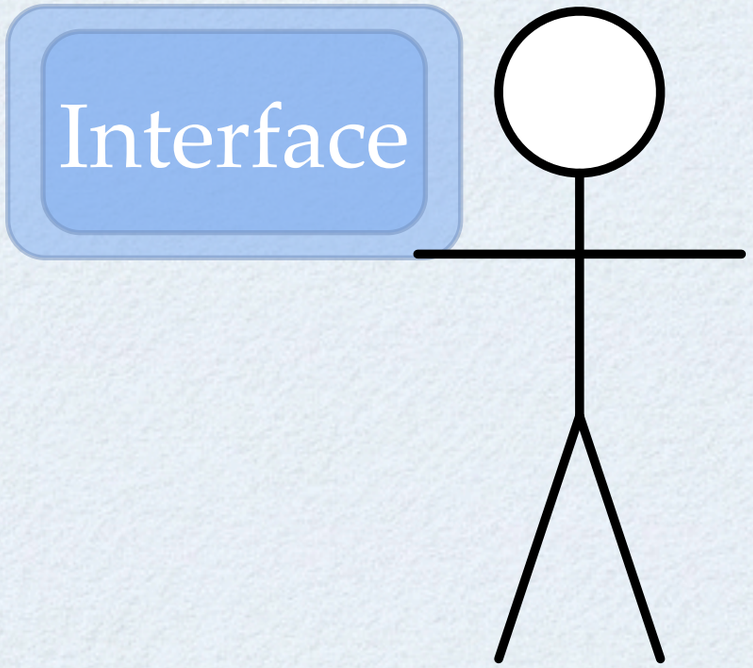

Interface

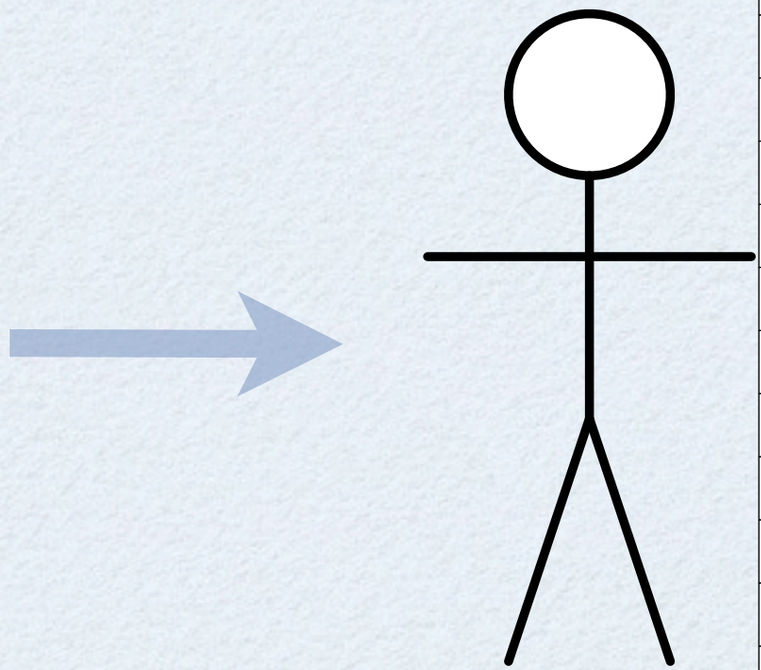


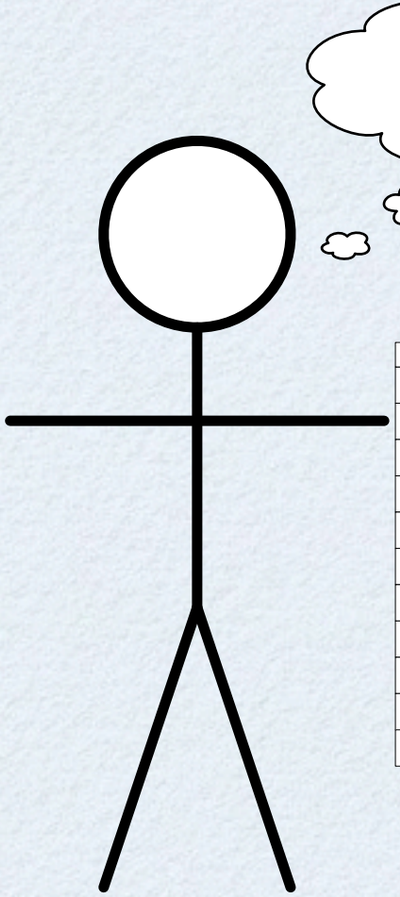


| Factor | Question |
|---|---|
| Abstraction Gradient | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| Consistency | Does this feature have consistent meaning? |
| Error-Proneness | Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world? |
| Hidden Dependencies | Are all dependencies clear with this feature? Could local changes have confusing global effects? |
| Premature Commitment | By using this language feature, do programmers have to make decisions before they have the information they need? |
| Progressive Evaluation | Can a partially-complete application of this feature provide feedback on “How am I doing”? |
| Viscosity | How much effort is required to perform a single change involving this feature? |
| Flexibility & Efficiency | Is this feature effective across the gradient between novice and expert programmers? |
| Aesthetic Design | Feature notation should be simple and concise. |
| Error Recovery | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| Documentation | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |

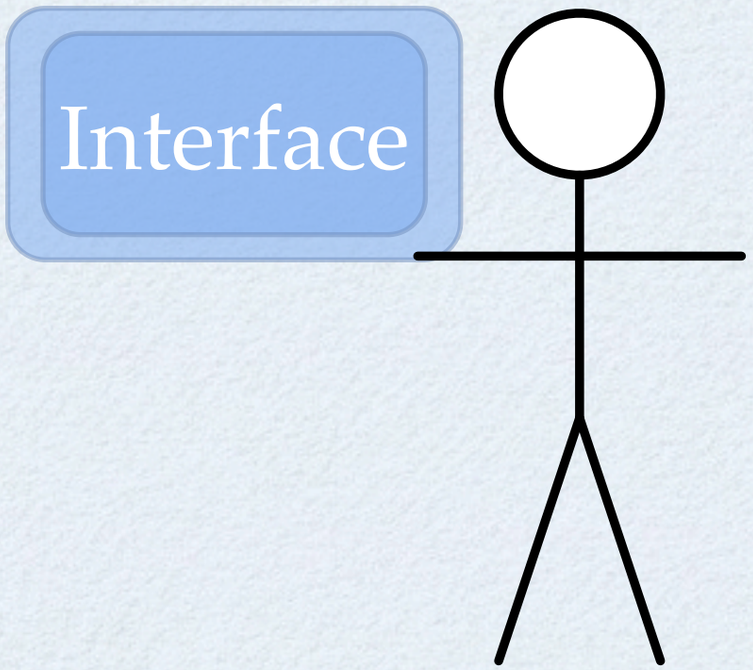


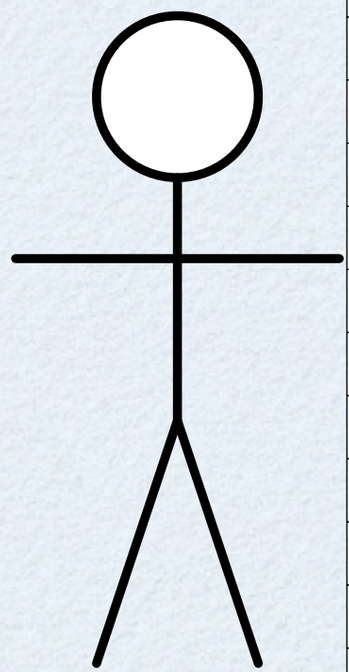



| Factor | Question |
|---|---|
| Abstraction Gradient | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| Consistency | Does this feature have consistent meaning? |
| Error-Proneness | Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world? |
| Hidden Dependencies | Are all dependencies clear with this feature? Could local changes have confusing global effects? |
| Premature Commitment | By using this language feature, do programmers have to make decisions before they have the information they need? |
| Progressive Evaluation | Can a partially-complete application of this feature provide feedback on “How am I doing”? |
| Viscosity | How much effort is required to perform a single change involving this feature? |
| Flexibility & Efficiency | Is this feature effective across the gradient between novice and expert programmers? |
| Aesthetic Design | Feature notation should be simple and concise. |
| Error Recovery | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| Documentation | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |

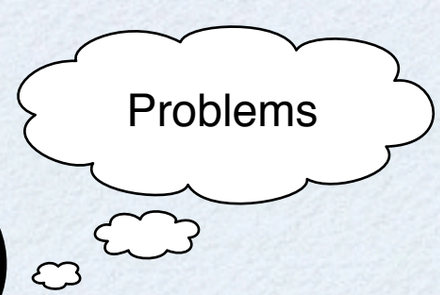


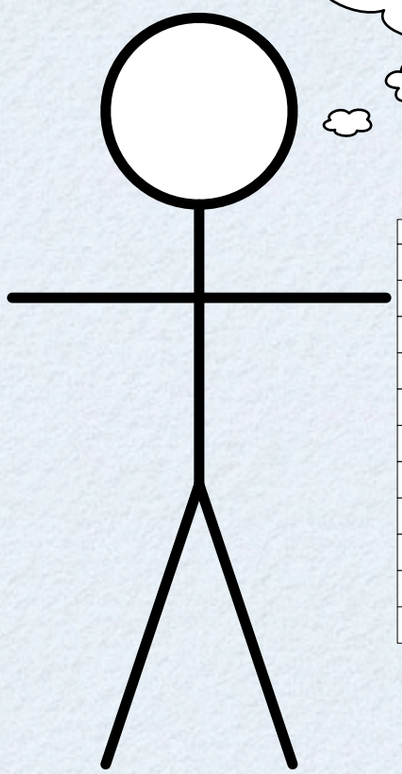

| Factor | Question |
|---|---|
| Abstraction Gradient | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| Consistency | Does this feature have consistent meaning? |
| Error-Proneness | Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world? |
| Hidden Dependencies | Are all dependencies clear with this feature? Could local changes have confusing global effects? |
| Premature Commitment | By using this language feature, do programmers have to make decisions before they have the information they need? |
| Progressive Evaluation | Can a partially-complete application of this feature provide feedback on “How am I doing”? |
| Viscosity | How much effort is required to perform a single change involving this feature? |
| Flexibility & Efficiency | Is this feature effective across the gradient between novice and expert programmers? |
| Aesthetic Design | Feature notation should be simple and concise. |
| Error Recovery | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| Documentation | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |



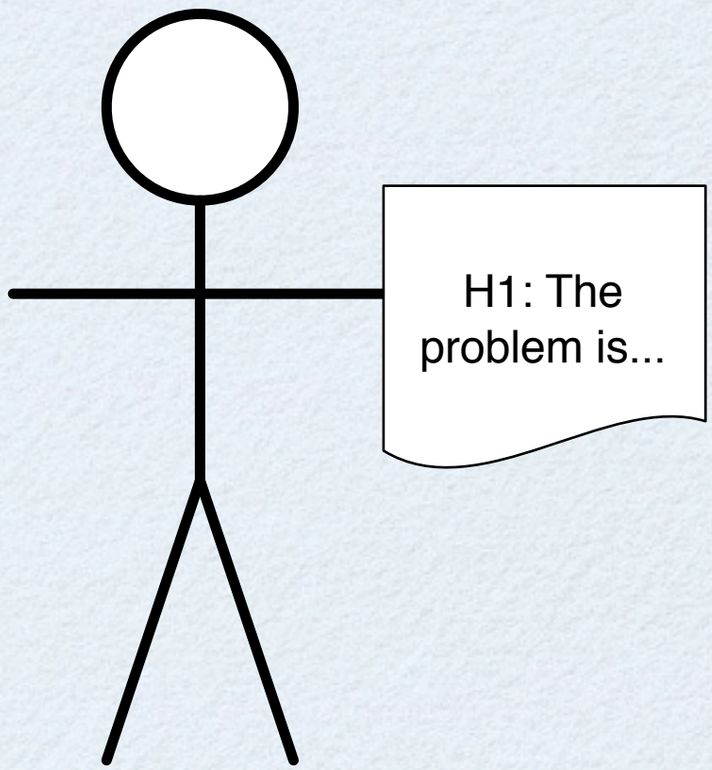



| Factor | Question |
|---|---|
| Abstraction Gradient | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| Consistency | Does this feature have consistent meaning? |
| Error-Proneness | Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world? |
| Hidden Dependencies | Are all dependencies clear with this feature? Could local changes have confusing global effects? |
| Premature Commitment | By using this language feature, do programmers have to make decisions before they have the information they need? |
| Progressive Evaluation | Can a partially-complete application of this feature provide feedback on “How am I doing”? |
| Viscosity | How much effort is required to perform a single change involving this feature? |
| Flexibility & Efficiency | Is this feature effective across the gradient between novice and expert programmers? |
| Aesthetic Design | Feature notation should be simple and concise. |
| Error Recovery | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| Documentation | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |



| Factor | Question |
|---|---|
| Abstraction Gradient | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| Consistency | Does this feature have consistent meaning? |
| Error-Proneness | Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world? |
| Hidden Dependencies | Are all dependencies clear with this feature? Could local changes have confusing global effects? |
| Premature Commitment | By using this language feature, do programmers have to make decisions before they have the information they need? |
| Progressive Evaluation | Can a partially-complete application of this feature provide feedback on “How am I doing”? |
| Viscosity | How much effort is required to perform a single change involving this feature? |
| Flexibility & Efficiency | Is this feature effective across the gradient between novice and expert programmers? |
| Aesthetic Design | Feature notation should be simple and concise. |
| Error Recovery | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| Documentation | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |



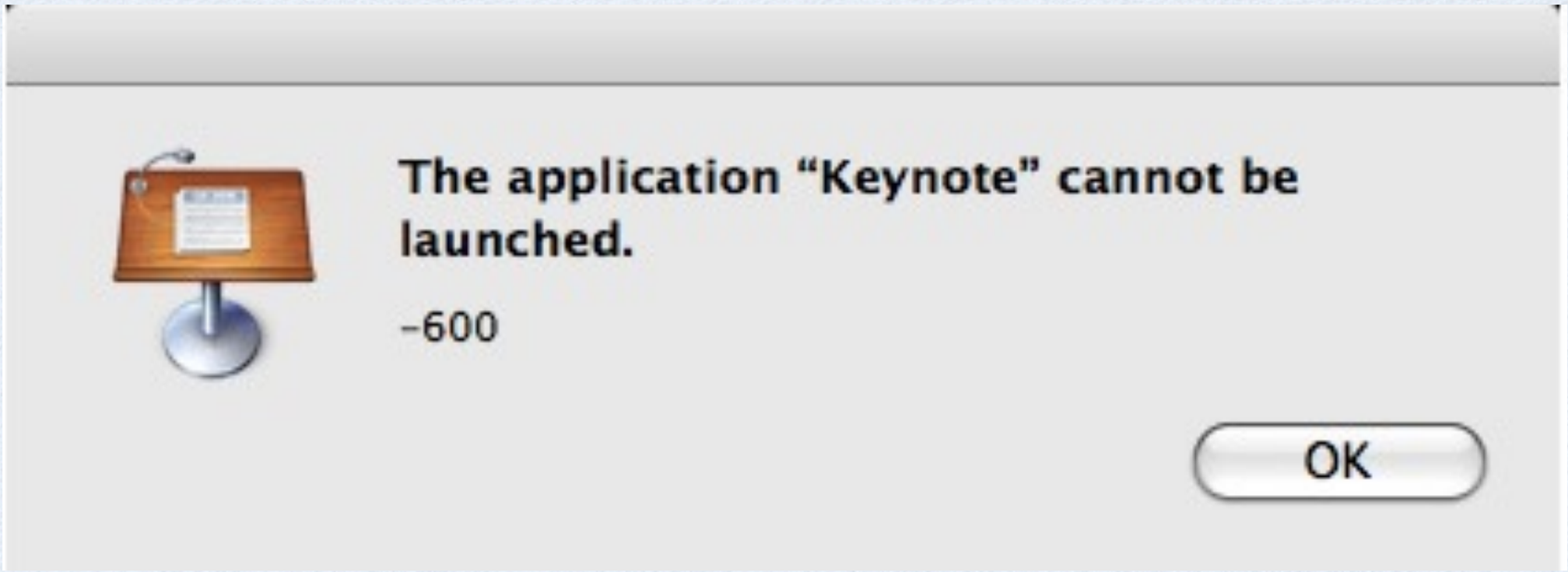
# HEURISTIC EVALUATION

- low-cost usability technique
- rate an interface along a set of guidelines "heuristics"
  - 0-4 scale
  - have to explain problems!

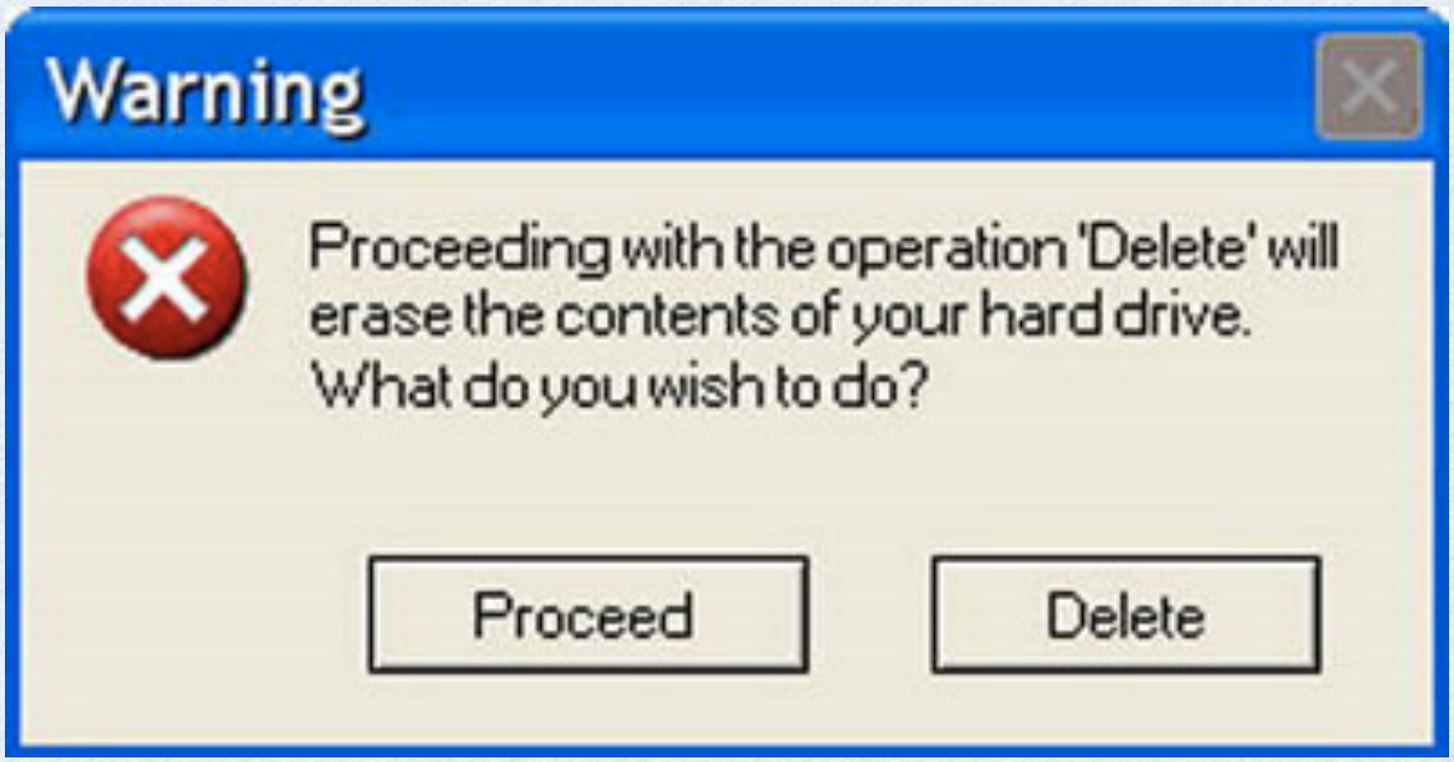
# EXAMPLE EVALUATION

- H: **Help users recognize, diagnose, and recover from errors.** Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

# USE A USER INTERFACE



# USE A USER INTERFACE



# Example Evaluation

- H: **Help users recognize, diagnose, and recover from errors.** Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
  - 3 (Major usability problem): Error message contains confusing text which does not make it clear what to do.

# Applications of Heuristic Evaluation

- Interfaces
- Games
- Ambient Displays
- ...

# YIELD ANNOTATIONS SPECIFY THREAD INTERFERENCE

```
lock(m);
while(x>0){
   unlock(m);
   lock(m);
}
assert x==0;
unlock(m);
```

# YIELD ANNOTATIONS SPECIFY THREAD INTERFERENCE

```
lock(m);
while(x>0){
   unlock(m);
   yield;
   lock(m);
}
assert x==0;
unlock(m);
```

# YIELD ANNOTATIONS

```
public class YieldExample {
    int x;
    public void foo() {
        float tmp1 = x;
        /* yield; */
        float tmp2 = x;
    }
    public void bar() {
        x = 3;
    }
}
```

- document point of thread interference
- no change in program behaviour

# ATOMIC ANNOTATIONS

```
public class AtomicExample {
    int x;
    public void foo() {
        atomic { float tmp1 = x; }
        atomic { float tmp2 = x; }
    }
    atomic public void bar() {
        x = 3;
    }
}
```

- *as if* no internal thread interference
- no change in program behaviour

# Developing heuristics for language features

- Merge, clarify, and trim:
  - Nielsen's 10 heuristics
    - 13 Cognitive Dimensions tradeoffs
- ... 11 language feature heuristics

| | |
|---|---|
| **Abstraction Gradient** | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| **Consistency** | Does this feature have consistent meaning? |
| **Error-Proneness** | Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world? |
| **Hidden Dependencies** | Are all dependencies clear with this feature? Could local changes have confusing global effects? |
| **Premature Commitment** | By using this language feature, do programmers have to make decisions before they have the information they need? |
| **Progressive Evaluation** | Can a partially-complete application of this feature provide feedback on “How am I doing”? |
| **Viscosity** | How much effort is required to perform a single change involving this feature? |
| **Flexibility & Efficiency** | Is this feature effective across the gradient between novice and expert programmers? |
| **Aesthetic Design** | Feature notation should be simple and concise. |
| **Error Recovery** | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| **Documentation** | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |

| | |
|---|---|
| **Abstraction Gradient** | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| **Consistency** | Does this feature have consistent meaning? |
| **Error-Proneness** | Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world? |
| **Dep** | |
| **Pr Co** | ions |
| **Pro Ev** | back |
| **Vi** | this |
| **Flexibility & Efficiency** | Is this feature effective across the gradient between novice and expert programmers? |
| **Aesthetic Design** | Feature notation should be simple and concise. |
| **Error Recovery** | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| **Documentation** | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |

**Hidden Dependencies:**

Are all dependencies clear with this feature? Could local changes have confusing global effects?

| | |
|---|---|
| **Abstraction Gradient** | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| **Consistency** | Does this feature have consistent meaning? |
| **Pr** | Does the notation for the language feature induce “careless |
| **Depe** | |
| **Pr Co** | |
| **Pro Eva** | |
| **Vi** | |
| **Flex Efficiency** | expert programmers. |
| **Aesthetic Design** | Feature notation should be simple and concise. |
| **Error Recovery** | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| **Documentation** | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |

**Error-Proneness:**

Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world?

| | |
|---|---|
| **Abstraction Gradient** | Does this feature adequately cover the gradient between minimum and maximum levels of abstraction in a program? (e.g. the gradient from low-level code details to interface-level details) |
| **Consistency** | Does this feature have consistent meaning? |
| **Error-Proneness** | Does the notation for the language feature induce “careless mistakes”? Is there a match between notation in the system and how similar language is used in the real world? |
| **Dep** | |
| **Pr Co** | ions |
| **Pro Eva** | back |
| **vi** | this |
| **Flexibility & Efficiency** | Is this feature effective across the gradient between novice and expert programmers? |
| **Aesthetic Design** | Feature notation should be simple and concise. |
| **Error Recovery** | If there is an error in language feature usage, are precise, constructive error messages in plain language presented? |
| **Documentation** | Even though it is better if the language feature can be used without documentation, it may be necessary to provide help and documentation. Documentation should be concise, concrete, and relevant. |

**Progressive Evaluation:**

Can a partially-complete application of this feature provide feedback on “How am I doing”?

# Running the Heuristic Evaluation

- 5 evaluators
  - computer science grad students
  - research on “yield” and “atomic”
- 15-20 min per evaluation

# RESULTS

- 5 problems identified with both
  - 2 not well discussed in literature (new)
- 7 problems identified with yield
  - 5 new
- 5 problems identified with atomic
  - 2 new
- Severity rating not helpful

# BOTH

- Tool feedback must be useful
- Word choice
  - *"Yield is an overloaded word; not self-documenting"*
- Misuse may lead to strange results
  - *"It is unclear how bugs related to inadequate yield annotations will actually manifest."*

# BOTH

- Documentation
- Interaction with evolving code
  - How do atomic blocks evolve over time?

# YIELD

- False positives or false negatives
- Ignoring annotations
- False sense of security
- Too many yields

# YIELD

- Local/global tension
  - *"Yields are specific to a line number; but how do you reason about methods that may contain yield annotations inside without looking at the code?"*
- Yield is too minimal
  - *"Which variables are affected by yield points?"*

# Atomic

- Bimodal reasoning
- Syntactic blocks limited
- Global impact of adding atomic
- Difficult to add well
- Class level

  - *"Does an atomic interface mean a safe interface?"*

# Participant Feedback

- All participants strongly agree:
  - either helped find new problems and gave new perspective, or clarified thinking
  - useful methodology for their research
- Lot of interest from PL community

# FUTURE WORK

- Clarify confusing heuristics
  - (abstraction gradient)
- Suggest additional heuristics
- Refine against usability problems with language features
- Evaluate against feature with set of known usability problems