Designing Useful Tools for Developers

Thomas D. LaToza and Brad A. Myers

Natural Programming

institute for SOFTWARE RESEARCH

Carnegie Mellon School of Computer Science

Helping developers bloop

You designed a **tool** to help developers *bloop*!
(automatic blooping!)

To **evaluate** your tool, you did a user study
your tool helped participants work significantly faster!

But is your tool useful?

Some questions about your results:

- Does the result **generalize**?
  - Would they be the same for different tasks, code, or participants?

- What does it mean for developers in the **field**?
  - How frequently do developers do these tasks?
  - Have they already found a more effective strategy?

- Did a lab study
  - Still doubts!
  - How can you possibly convince someone it's useful?

Useful tools

What does it mean for a tool to be **useful**?

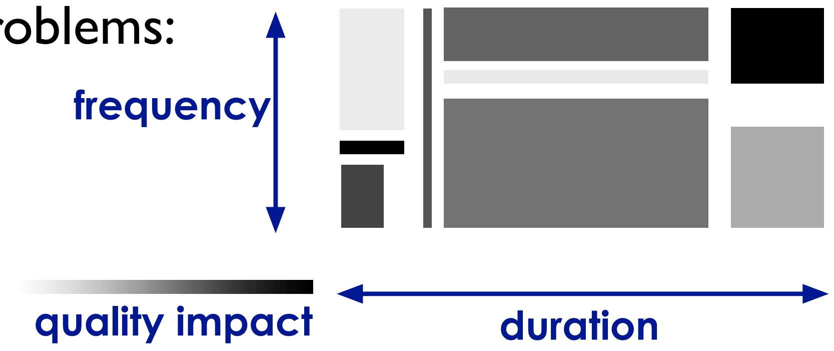
- Not a question of philosophy, mathematics, or esthetics
- **Scientific**, falsifiable question about nature
- If developers adopt a tool, is their work faster or better?
- But you can't **prove** usefulness

Usefulness is a **theory** of a tool's effect on work and **evidence** to support

- Studies develop theory and provide evidence

Useful tools **solve important** problems

- Important problems:



- Solve: supports how developers work (**mechanism**)

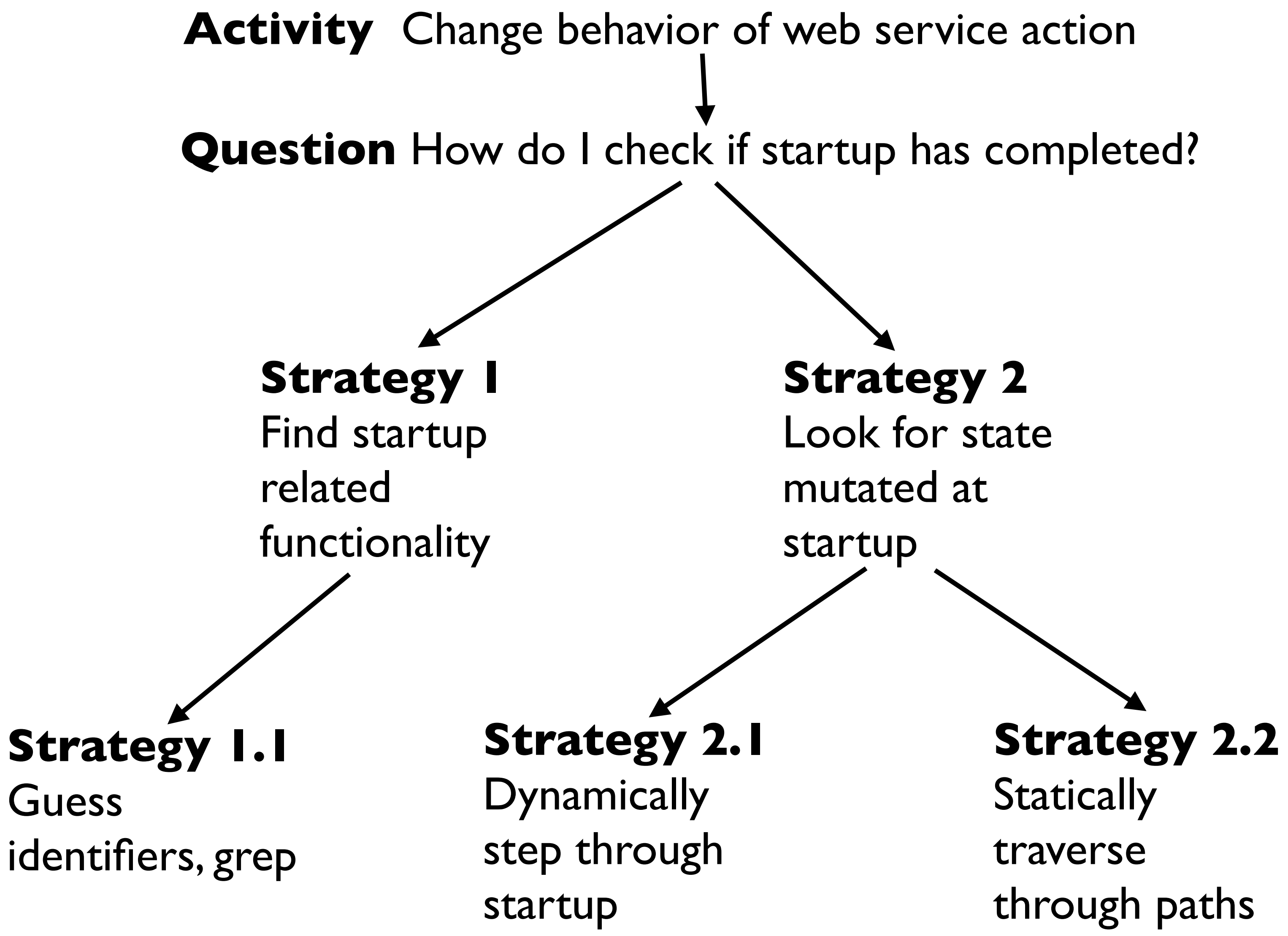
Supporting developers' work

Work consists of **tasks**: activities, strategies or steps

Tasks have a **goal** - **question** to answer, thing to **do**

$$\begin{tikzcd}
 & \text{work} \arrow[d] \\
\text{high level goal} & \text{activities} \arrow[d] \arrow[loop right] \\
\text{specific approach} & \text{strategies} \arrow[d] \\
\text{individual action} & \text{steps} \arrow[u, bend right]
\end{tikzcd}$$

Supporting developers' work


Activity Change behavior of web service action
Question How do I check if startup has completed?
Strategy 1 Find startup related functionality
Strategy 2 Look for state mutated at startup
Strategy 1.1 Guess identifiers, grep
Strategy 2.1 Dynamically step through startup
Strategy 2.2 Statically traverse through paths

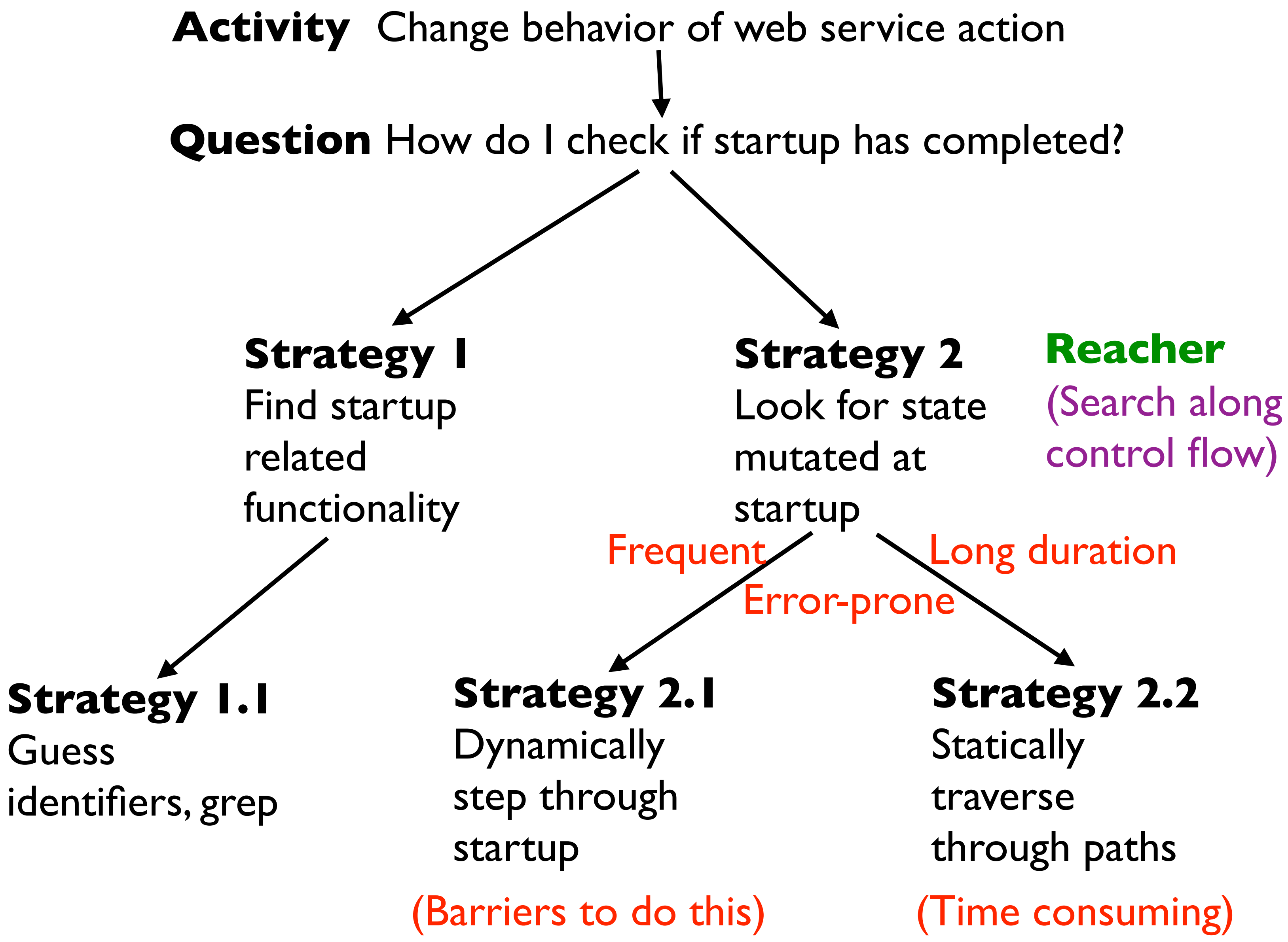
Supporting developers' work

Tools support **strategies**

Useful tools support **important** strategies

Solve **problems** developers have using them

Supporting developers' work



Activity Change behavior of web service action

Question How do I check if startup has completed?

Strategy 1
Find startup related functionality

Strategy 2
Look for state mutated at startup

Reacher
(Search along control flow)

Frequent

Error-prone

Long duration

Strategy 1.1
Guess identifiers, grep

Strategy 2.1
Dynamically step through startup

(Barriers to do this)

Strategy 2.2
Statically traverse through paths

(Time consuming)

A theory of Reacher’s usefulness

Activity Debugging

Activity Implications of changes

Reacher

Strategy Search along control flow

Frequent Long duration Error-prone

Automates

Strategy Static traversal

Problem Must guess which paths to traverse

Solution Enter searches

Problem Paths may be infeasible

Solution Fast feasible path analysis

The role of empirical studies

Understand problems

Design a solution

Evaluate the solution

Designing useful tools from **data**!

Understand problems

(importance)

Solve **important** problems
=> exploratory studies identify and describe problems

Design a solution

(mechanism)

Make sure you **solve** problems
=> lightweight evaluation studies test design ideas early

Evaluate the solution

(mechanism)

Evaluate **how** the tool supports work
=> lab studies with quantitative and qualitative measures

Solve an **important** problem

Adopting a tool is an **investment** for developers

Developers need to be convinced problems warrant investment

Developers believe academic tools solve **unimportant** problems [Aranda+11]

Copy and paste reuse

Copy and paste reuse creates **clones**
- developers copy small snippets of code to create clones
- future changes must then modify all clones

If only developers could find these, they'd refactor them!
- many clone detectors designed to do this (c.f., CCFinder)

But.... hard to convince developers in field to adopt.
- is it a problem of **finding** or of **expressiveness**? [Toomim+04]
- how important is this problem?

But.... commercial clone detectors have been successful!
- **entire codebase** also copied for versions, configurations, releases
- developers must fix defect in all copies
- missing a defect which is important enough to fix is a big problem
- commercial clone detector primarily used for this [Pattern Insight]

Identifying **important** problems

How do you identify and understand important problems?

Using exploratory studies!

- Interviews - **ask** about activities, problems, strategies, questions, ...
- Contextual inquiries - watch + interview developers
- Direct observations - watch developers
- Indirect observations - logging, code, changes, emails, bugs, forums, ...
- Surveys - frequency data, examples, find infrequent problems

Make sure you **solve** the problem

How **much** of the problem does your tool address?

What strategy is supported?

How many of the steps are supported?

Automated debuggers

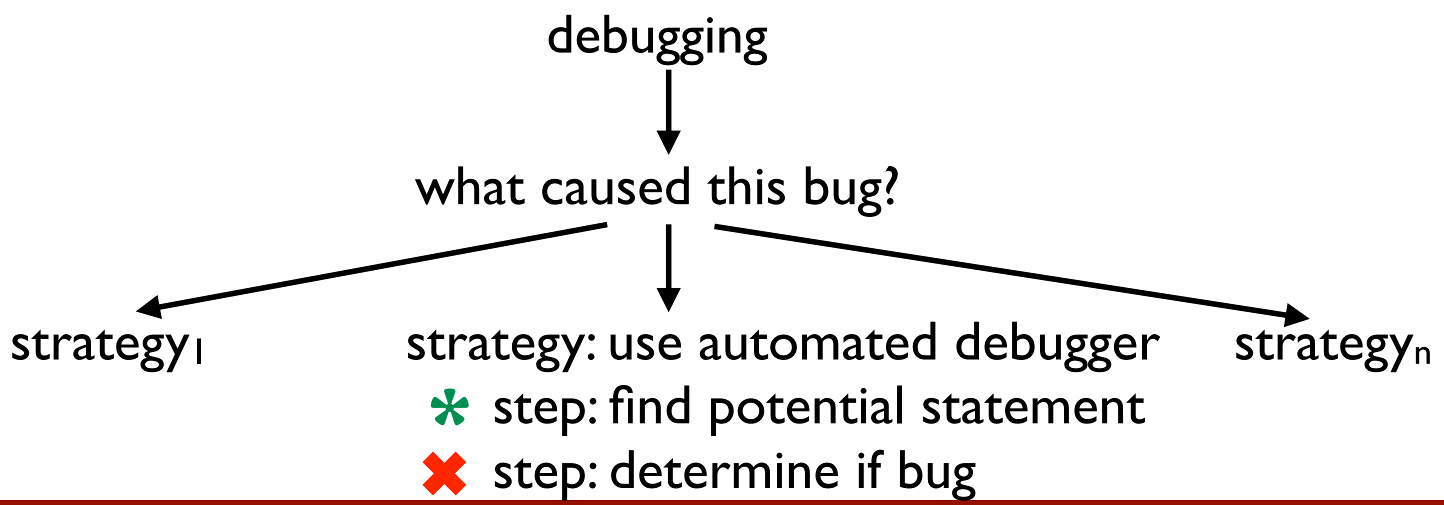
Predict faulty statement - Tarantula [Jones+02]

Provide ranked **list** of possibly buggy statements

Does this **help** developers debug?

- Lab study of automated debugging tools [Parnin+11]
- Helped best performers debug easy bug
- Did not help debug hard bug
  - Not even if rank artificially boosted (from 83 to 16)

Tools assumed perfect bug understanding - **not** true



Make sure you **solve** the problems

Understanding the problem

- What is the **higher** level question / goal your tool supports?
  - Not finding buggy statement, but understanding
- How does your tool support it?

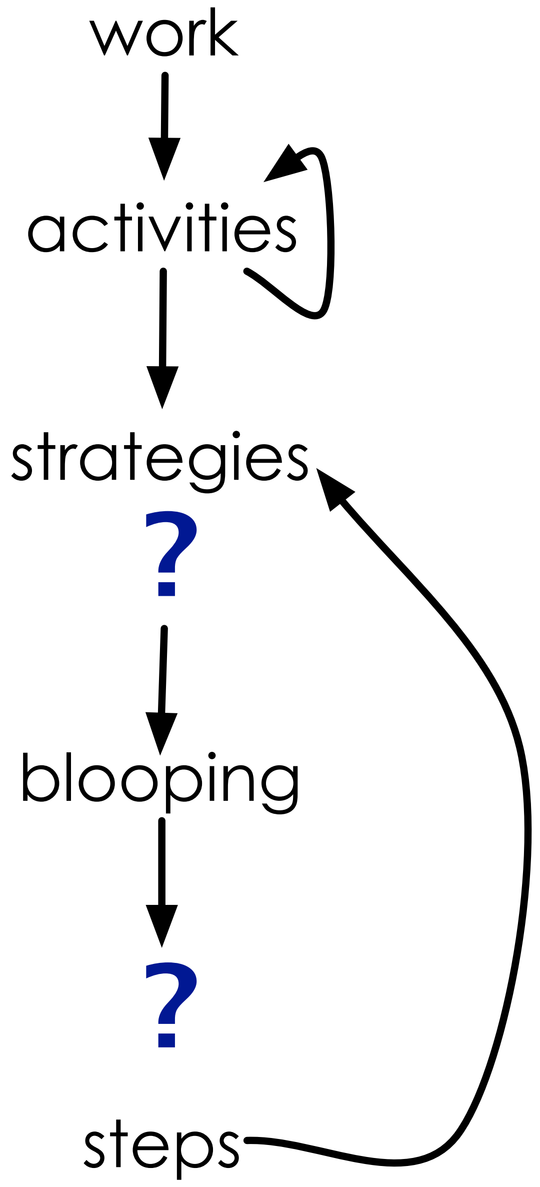
Lightweight evaluations - test ideas **early**

- Simulate a tool!
- Paper prototypes
  - Developer interacts with **screenshots**, narrates actions
  - Experimenter manually simulates the tool, showing next screen
- Wizard of oz - higher fidelity
  - Mockup interface built
  - Developer believes interacting with **real** tool
  - Behind the curtain, experimenter simulates tool

Evaluate **how** tool supports work

By what mechanism does it support work?

- What problems does it solve?
- What question(s) does it help answer?
- Or what does it help do?
- What strategies does it support?



Dynamic vs. static typing

Controversy over whether untyped/dynamically typed (e.g., Perl, Ruby) or statically typed (e.g., Java, C#) languages make developers more productive

- Do static types get in the way or provide important feedback?

Lab study! [Hanenberg09]

- Graduate students writing parser over 27 hrs
- Is untyped or typed variant faster?
- **Untyped** took 4- 42% less time

What does this **mean**?

- Would developers be more productive without types?
- Generalizability - when?
  - How is writing types slowing developers? Could this be improved?
  - Does the feedback ever help? How does this effect work?

Evaluate **how** tool supports work

Lab studies should include quantitative **and** qualitative measures

Quantitative results
- What are the effects on task duration and quality?
- Existence proof: **can** it solve problems?

Qualitative results
- **How** does it solve problems? (mechanism)
- In what ways does it help answer questions? Which ones?
- How does it change how developers work?

Usefulness hypotheses helps **design** study
- What code, tasks, materials will focus on most interesting situations?

Conclusions

Useful tools **solve important** problems

Theory of how tool supports work, evidence supporting theory

Gathering **data** before, during, and after tool design

Helps identify problems, ensure you solve problems, understand how it helps

Wide range of techniques for gathering data

Questions?

Acknowledgements

grant CCF-0811610

Contact

Thomas LaToza
tlatoza@cs.cmu.edu