

InFix: Automatically Repairing Novice Program Inputs

Madeline Endres



InFix: Automatically Repairing Novice Program Inputs

Madeline Endres



Recently accepted for
publication at *Automated
Software Engineering*, a top
Software Engineering
conference (20%
acceptance rate)!

InFix: A Brief Overview

- **Motivation:** Non-traditional novice programmers are growing in numbers. We want to help them **debug and understand** their programs. Many novice bugs relate to input.
 - Both novice bugs in general and input-related bugs in particular, are underserved by current research efforts
- **Method:** Adapt techniques from **search-based automated program repair** to fix novice input-related bugs (InFix)
- **Evaluation:** An **empirical evaluation** of InFix on 25,000+ programs and an IRB-approved **human study**



The online Python Tutor interpreter currently has 60,000 users per month

Many People Want to Learn to Code

Without traditional classroom support

How do Codecademy's 45 million users learn to code?

FULL-TIME COURSES



only
1/3
took a
full-time
course

ONLINE COURSES



35%
said online courses
were their primary
method for learning



1/2
have never
taken a
university
course

GeekWire NEWS ▾ JOBS EVENTS ▾ RESOURCES ▾ ABOUT ▾ □ □ □ □ □ Search

Coding bootcamps see huge enrollment increase



Our Dataset: Python Tutor

Python Tutor is a free online **interpreter**. It helps **novices visualize** code execution

In the past 4 years, it had over 200,000 unique users

33% of all user interactions involved a program with a call to `input()`

25,000+ input-related bugs: instances the user fixed an error by modifying only the input data



Our Dataset: Python Tutor

Python Tutor is a free online **interpreter**. It helps **novices visualize** code execution

Key Idea:

Input-related bugs are **common** for novice programmers, but they are **overlooked** by current state-of-the-art teaching tools

25,000+ input-related bugs. Instances the user fixed an error by modifying only the input data

Write code in Python 3.6

```
1 u = 42
2 x = float(input())
3 print(x * math.e / 2)
```

Help improve this tool by completing a [short user survey](#)

Please wait ... executing (takes up to 10 seconds)

Live Programming Mode

Input Repair: Ideal Solution vs. Current

Ideal Solution Element

- Provide repairs that are **actually helpful** for novices (high quality)
- **Find repairs quickly** for live feedback on **diverse programs**
- Include support for **input-related** bugs

Current State of the Art

- Repair tools (e.g. GenProg) are **confusing** for novices (Yi et al. 2017)
- Repair optimizations require large course settings with **common assignments** (Ahmed et al. 2018)
- Input repair is only proposed for **security** bugs (Long et al. 2012)



Talk Outline



1. Python Tutor and Input-Related Errors
 - a. What do input-related errors look like?
2. The InFix **algorithm**: Automatically fixing input-related bugs
 - a. Algorithmic design decisions
 - b. Our Python specific InFix implementation
3. **Evaluation**: Does InFix work? (spoiler, yes!)
 - a. Both a comprehensive evaluation based on program semantics and also a controlled human study to assess repair quality
 - b. An empirical evaluation on 25,000+ input error scenarios

Simple **Syntactic** Input-Related Error

The Code

```
u = 42
```

```
x = float(input())
```

```
print(x * math.e / 2)
```



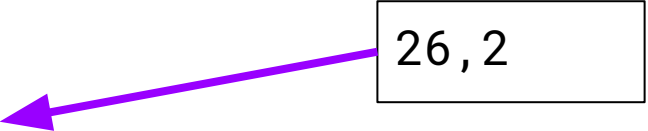
Simple **Syntactic** Input-Related Error

The Code

```
u = 42
x = float(input())
print(x * math.e / 2)
```

The Student Input

26, 2



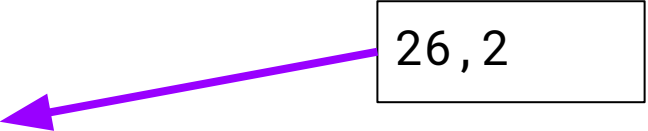
Simple **Syntactic** Input-Related Error

The Code

```
u = 42
x = float(input())
print(x * math.e / 2)
```

The Student Input

26,2



The Error Message

```
ValueError: could not convert
string to float: '26,2'
```




Simple **Syntactic** Input-Related Error

The Code

```
u = 42
x = float(input())
print(x * math.e / 2)
```


The Student Input

26,2



Possible Repair

29.2



The Error Message

```
ValueError: could not convert
string to float: '26,2'
```



Syntactic Input Bugs: A Common Pattern

The Student Input

26,2

The Error Message

```
ValueError: could not  
convert string to  
float: '26,2'
```

Possible Repair

29.2



The Student Input

\$1.50

The Error Message

```
ValueError: could not  
convert string to  
float: '$1.50'
```

Possible Repair

1.50

The Student Input

math.py/6

The Error Message

```
ValueError: could not  
convert string to  
float: 'math.py/6'
```

Possible Repair

3.14

More Complex **Semantic** Input-Related Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```



More Complex **Semantic** Input-Related Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}
```

```
for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]
```

```
print(c_array)
```

Key Takeaways

- input_b must be at least as long as input_a
- input_c must consist **only of characters** that are in input_a



More Complex **Semantic** Input-Related Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```



More Complex **Semantic** Input-Related Error

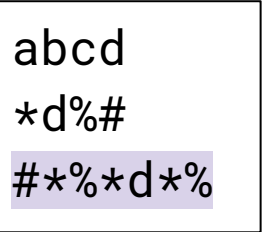
The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

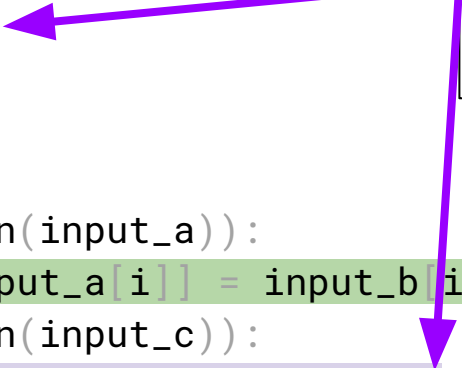
for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```

The Student Input



```
abcd
*d%#
#*%*d*%
```



More Complex **Semantic** Input-Related Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```

The Student Input

```
abcd
*d%#
#*%*d*%
```

The Error Message

```
KeyError: '#'
```



More Complex **Semantic** Input-Related Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}
```

```
for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```

The Student Input

```
abcd
*d%#
#*%*d*%
```

Possible Repair

```
abcd
*d%#
abcd
```

The Error Message

```
KeyError: '#'
```



Input-Related Errors: Key Insights

Insight

- If novice repairs are generally **short**
- If many student programs with the same error message have **similar fixes**
- If student inputs are often **structurally complex** with interdependent input values but have **simple fixes**



Repair Algorithm Implication

- We can explore the **search space** of nearby edits to find fixes
- A small number of indicative **error-message templates** can increase search speed
- A **randomized** approach is surprisingly effective

InFix Algorithm: Formal Input and Output Properties

Inputs

- **Program P** : program code
- **Erroneous input I** : token_sequence
- **Error message M** : string
- **Error Message Template function T** : string \rightarrow Mutation
- **Set of general Mutations R** : set of Mutations
- **Max number of probes (iterations) N** : \mathbb{N}

Output

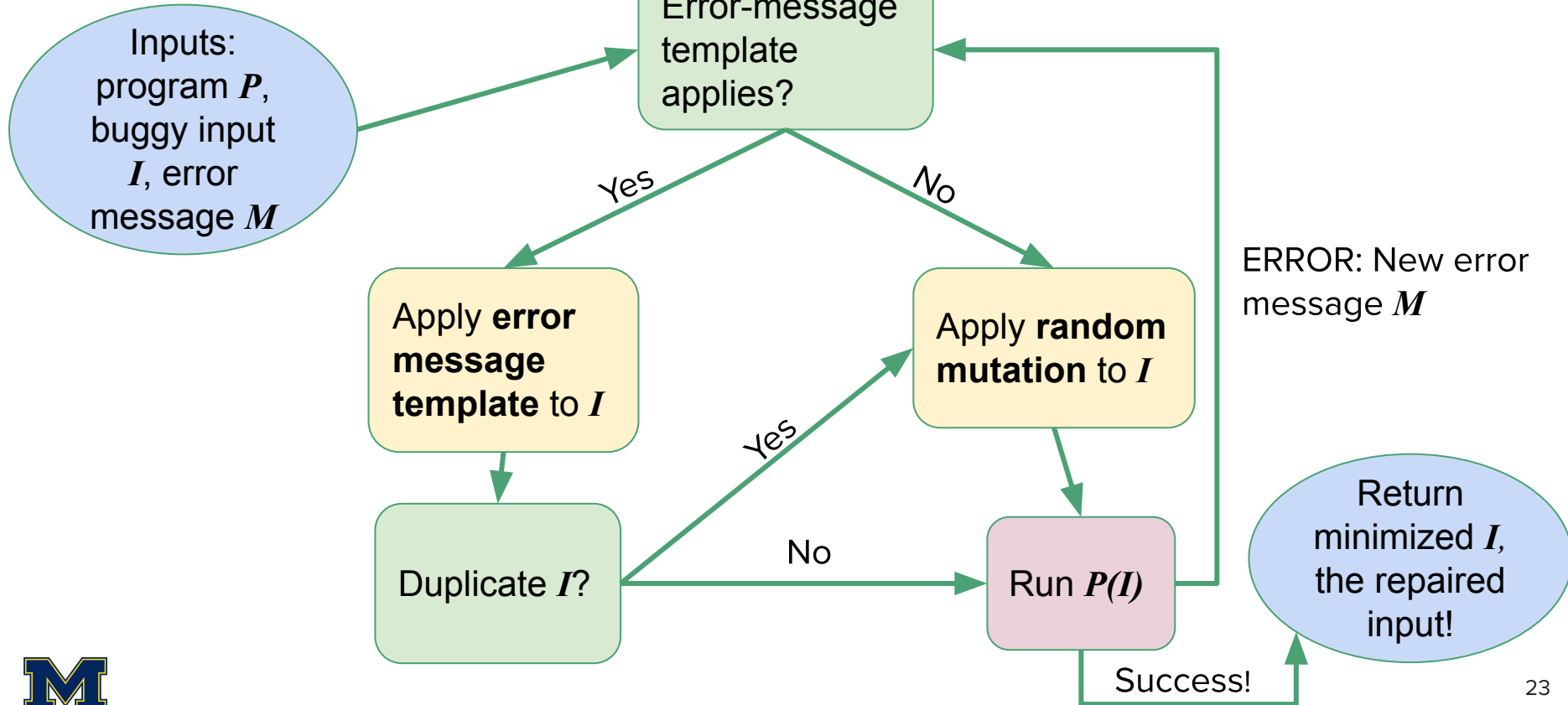
A repaired input that does not produce an error when run with program P

OR

TIMEOUT



InFix Algorithm



InFix Example Walkthrough: Semantic Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```

| Current Input | Error Message | Action |
|-------------------------|------------------|--------|
| abcd *d%# #*%*d*% | KeyError: '#' | |



InFix Example Walkthrough: Semantic Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```

| Current Input | Error Message | Action |
|-------------------------|------------------|--------------------------------------|
| abcd *d%# #*%*d*% | KeyError: '#' | Random Mutation: Remove random token |
| abcd *d%# | | |



InFix Example Walkthrough: Semantic Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```

| Current Input | Error Message | Action |
|-------------------------|------------------|--------------------------------------|
| abcd *d%# #*%*d*% | KeyError: '#' | Random Mutation: Remove random token |
| abcd *d%# | EOFError | |



InFix Example Walkthrough: Semantic Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```

| Current Input | Error Message | Action |
|-------------------------|------------------|---|
| abcd *d%# #*%*d*% | KeyError: '#' | Random Mutation: Remove random token |
| abcd *d%# | EOFError | Error Message template: Generate new token - either random or from bad input |
| abcd *d%# abcd | | |



InFix Example Walkthrough: Semantic Error

The Code

```
input_a = input()
input_b = input()
input_c = input()
c_array = []
dictionary = {}

for i in range(len(input_a)):
    dictionary[input_a[i]] = input_b[i]
for j in range(len(input_c)):
    c_array += dictionary[input_c[j]]

print(c_array)
```

| Current Input | Error Message | Action |
|-------------------------|------------------|---|
| abcd *d%# #*%*d*% | KeyError: '#' | Random Mutation: Remove random token |
| abcd *d%# | EOFError | Error Message template: Generate new token - either random or from bad input |
| abcd *d%# abcd | Success! | |



InFix Evaluation: Does it work?

Five Research Questions

1. How effective is InFix?
2. How high quality are InFix's repairs?
 - a. Empirical Evaluation
 - b. Human Study
3. *Are InFix's Design Assumptions Valid?*
4. *How sensitive is InFix to input parameters?*
5. *Does expertise affect InFix's helpfulness?*



InFix Evaluation

Five Research Questions

1. How effective is InFix?
2. How high quality are InFix's repairs?
 - a. Empirical Evaluation
 - b. Human Study
3. *Are InFix's Design Assumptions Valid?*
4. *How sensitive is InFix to input parameters?*
5. *Does expertise affect InFix's helpfulness?*

Benchmarks



InFix Evaluation

Five Research Questions

1. How effective is InFix?
2. How high quality are InFix's repairs?
 - a. Empirical Evaluation
 - b. Human Study
3. *Are InFix's Design Assumptions Valid?*
4. *How sensitive is InFix to input parameters?*
5. *Does expertise affect InFix's helpfulness?*

Benchmarks

1. Python Tutor Data Set



InFix Evaluation

| Year | Number of Input-Related Bugs |
|-------|------------------------------|
| 2015 | 1,640 |
| 2016 | 4,440 |
| 2017 | 6,949 |
| 2018 | 12,723 |
| Total | 25,995 |

Benchmarks

1. Python Tutor Data Set



InFix Evaluation

Five Research Questions

1. How effective is InFix?
2. How high quality are InFix's repairs?
 - a. Empirical Evaluation
 - b. Human Study
3. *Are InFix's Design Assumptions Valid?*
4. *How sensitive is InFix to input parameters?*
5. *Does expertise affect InFix's helpfulness?*

Benchmarks

1. Python Tutor Data Set
2. IRB-Approved Human Study



InFix Evaluation: Focused Research Questions

| Research Question | Evaluation Metric | Success Criterion |
|--|--|--|
| RQ1: How effective is InFix? | % Inputs repaired | >= 80% (Ahmed et al., 2018) |
| RQ2: How high quality are InFix's repairs? | Statement coverage | >= 75% (Tillmann et al., 2008) |
| | Human subjective assessment of quality | >= 75% the quality of human patches (Kim et al., 2013) |

RQ1: How Effective is InFix?

- 25,995 scenarios with **input-related** errors
 - Pulled from historical Python Tutor student data
- InFix Parameter Settings:
 - Max_Probes = 60
 - Parallel threads = 5
- Results: InFix repairs **94.5%** of scenarios in a median time of **0.88 seconds**
 - This exceeds our **success** criteria of $\geq 80\%$!
 - And is fast enough to provide **real time help** in the majority of cases!



RQ1: How Effective is InFix?

| Year | Input-Error Scenarios | | | Probes to Solve | | Time (sec) | |
|--------------|-----------------------|---------------|--------------|-----------------|-------------|-------------|-------------|
| | Total | Repaired | % | Med | Avg | Med | Avg |
| 2015 | 1,640 | 1,582 | 96.5 | 1 | 2.98 | 0.87 | 1.12 |
| 2016 | 4,440 | 4,683 | 94.8 | 2 | 3.23 | 0.88 | 1.16 |
| 2017 | 6,949 | 6,590 | 94.8 | 2 | 3.47 | 0.90 | 1.23 |
| 2018 | 12,723 | 11,947 | 93.9 | 2 | 3.70 | 0.88 | 1.28 |
| Total | 25,995 | 24,559 | 94.5% | 2 | 3.50 | 0.88 | 1.23 |



RQ1: How Effective is InFix?

| Year | Input-Error Scenarios | | | Probes to Solve | | Time (sec) | |
|--------------|-----------------------|---------------|--------------|-----------------|-------------|-------------|-------------|
| | Total | Repaired | % | Med | Avg | Med | Avg |
| 2015 | 1,640 | 1,582 | 96.5 | 1 | 2.98 | 0.87 | 1.12 |
| 2016 | 4,440 | 4,683 | 94.8 | 2 | 3.23 | 0.88 | 1.16 |
| 2017 | 6,949 | 6,590 | 94.8 | 2 | 3.47 | 0.90 | 1.23 |
| 2018 | 12,723 | 11,947 | 93.9 | 2 | 3.70 | 0.88 | 1.28 |
| Total | 25,995 | 24,559 | 94.5% | 2 | 3.50 | 0.88 | 1.23 |



RQ1: How Effective is InFix?

| Year | Input-Error Scenarios | | | Probes to Solve | | Time (sec) | |
|--------------|-----------------------|---------------|--------------|-----------------|-------------|-------------|-------------|
| | Total | Repaired | % | Med | Avg | Med | Avg |
| 2015 | 1,640 | 1,582 | 96.5 | 1 | 2.98 | 0.87 | 1.12 |
| 2016 | 4,440 | 4,683 | 94.8 | 2 | 3.23 | 0.88 | 1.16 |
| 2017 | 6,949 | 6,590 | 94.8 | 2 | 3.47 | 0.90 | 1.23 |
| 2018 | 12,723 | 11,947 | 93.9 | 2 | 3.70 | 0.88 | 1.28 |
| Total | 25,995 | 24,559 | 94.5% | 2 | 3.50 | 0.88 | 1.23 |



RQ1: How Effective is InFix?

| Year | Input-Error Scenarios | | | Probes to Solve | | Time (sec) | |
|--------------|-----------------------|---------------|--------------|-----------------|-------------|-------------|-------------|
| | Total | Repaired | % | Med | Avg | Med | Avg |
| 2015 | 1,640 | 1,582 | 96.5 | 1 | 2.98 | 0.87 | 1.12 |
| 2016 | 4,440 | 4,683 | 94.8 | 2 | 3.23 | 0.88 | 1.16 |
| 2017 | 6,949 | 6,590 | 94.8 | 2 | 3.47 | 0.90 | 1.23 |
| 2018 | 12,723 | 11,947 | 93.9 | 2 | 3.70 | 0.88 | 1.28 |
| Total | 25,995 | 24,559 | 94.5% | 2 | 3.50 | 0.88 | 1.23 |



RQ1: How Effective is InFix?

| Year | Input-Error Scenarios | | | Probes to Solve | | Time (sec) | |
|--------------|-----------------------|---------------|--------------|-----------------|-------------|-------------|-------------|
| | Total | Repaired | %Repaired | Med | Avg | Med | Avg |
| 2015 | 1,640 | 1,582 | 96.5 | 1 | 2.98 | 0.87 | 1.12 |
| 2016 | 4,440 | 4,683 | 94.8 | 2 | 3.23 | 0.88 | 1.16 |
| 2017 | 6,949 | 6,590 | 94.8 | 2 | 3.47 | 0.90 | 1.23 |
| 2018 | 12,723 | 11,947 | 93.9 | 2 | 3.70 | 0.88 | 1.28 |
| Total | 25,995 | 24,559 | 94.5% | 2 | 3.50 | 0.88 | 1.23 |



RQ1: How Effective is InFix?



InFix **repairs 94.5%** of input-related scenarios in a median **0.88 seconds**

This is **high** compared to that achieved by related work
(Ahmed et al., 2018 = 80%)

RQ2: What is the Quality of InFix's Repairs?

- In Automated Program Repair, **patch quality** is a **current research focus**
 - One of two "**ongoing challenges**" mentioned in *It Does what you Say, not what you Mean: Lessons from a Decade of Program Repair* (ICSE Most Influential Paper Keynote, 2019)
- We will assess InFix's repair quality from multiple angles:
 1. A broad **semantics-based** approach using **code coverage** on nearly 12,000 programs
 2. A **human evaluation** with 97 participants on 60 randomly-chosen stimuli



RQ2: Why Code Coverage?

- Statement coverage pros:
 - **Scalable** for large evaluation
 - **Directly comparable** to previous work
- Coverage can indicate if InFix commonly produces non-interesting inputs that bypass large blocks of code (eg. empty list)
- Coverage is a coarse approximation of **programmer intent**



RQ2: Code Coverage approach

Study Setup

- For the 11,947 repaired programs from the Python Tutor 2018 data, collected **statement coverage** for:
 - a. Inputs generated by InFix
 - b. Historical student created input repair

Results

- InFix repairs achieve a median 83.3% coverage
- Student repairs achieve a median 90.3% coverage
- InFix is within 7% of the student repair coverage upper bound
- **InFix coverage is high** → similar to PEX (71%-95%, 2008) and higher than KATCH (52%, 2013)



RQ2: IRB- Approved Human Evaluation Set-Up

- Randomly selected 60 programs from the Python Tutor Data
- For each, made two stimuli: One with InFix's repair, and one with the historical student repair
- Humans describe the cause of the bug
- Humans judge the **quality** of the suggested input repair (Likert)
 - Likert scale questions help us measure the **subjective human experience**
- Humans judge the helpfulness of the suggested input repair (Likert)
- **97 participants:** 24 Michigan Students, 73 MTurk Workers
 - Manually verified response quality using participant's answer to the bug description question



RQ2: IRB- Approved Human Evaluation Set-Up

Python Program

```
1 | n, m= (int(i) for i in input().split())
2 | a = [[0 for j in range(m)] for i in range(n)]
3 | print(a)
```

Bug Revealing Input and Error Message

Bug Revealing Input

9 5 3

Error Output

Traceback (most recent call last):

File "temp2018.py", line 1, in <module>

n, m= (int(i) for i in input().split())

ValueError: too many values to unpack (expected 2)

Suggested Input Repair

Repaired Input

9 5

Output Produced by Repair

```
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```



RQ2: Human Evaluation Analysis

- InFix achieves **96% the quality** of historical student repairs ($p = 0.047$)
- Statistical significance determined using two-tailed **Mann-Whitney U Test**
 - Student's t-test not applicable as distribution was not Normal
 - Mann-Whitney is **non-parametric** and requires **independent samples**
- **This quality is high** compared to other repair quality studies
 - PAR patches were **75.5%** as acceptable as human generated patches (2013)
 - **46.1%** of Prophet patches were manually found to be correct (2016)





InFix repairs are of **high quality**

On average, InFix achieves **83.3% coverage**, 90.2% the coverage of student repairs

97 study participants found InFix to be **96% the quality** ($p = 0.047$) of human repairs

InFix: Effective, High Quality, and Supports Learners at Scale

In summary, my research contributions are:

1. **InFix**, a novel **template-based search algorithm** for repairing input-related bugs
 - a. Identifying **input data** as an important source of **novice programmer** bugs
 - b. Characterizing **common novice input patterns** for Python, resulting in **error message templates** and **general mutation** operators
2. An **implementation** and **evaluation** of InFix
 - a. **Fixes 94.5%** of 25,995 input-related errors in a median of **0.88 seconds**
 - b. Produces repairs that achieve within **7% the coverage** and also **96% of the quality** of student repairs ($p = 0.047$)



BONUS



Bonus Slide: Statistics 1

- Student's T-Test
 - **Parametric** → Assumes a normal distribution
 - Has both independent and paired forms
- Mann-Whitney U-Test
 - **Non-parametric**
 - Requires **independent** samples
- Wilcoxon signed-rank test
 - **Non-parametric**
 - Requires **paired** samples



Bonus Slide: Statistics 2

- Fleiss' Kappa
 - Test of interrater reliability
 - Assumes fixed number of annotators are giving categorical ratings
- P-value
 - The probability that the data is this skewed
 - Assumes that the null hypothesis is true
- P-Hacking / Data dredging
 - Using multiple statistical tests and multiple hypothesis, but only reporting significant results
 - related to spurious correlations, multiple comparisons, and false discovery rate



Error-Message templates for Python: T

| Error Message | Associated Input Mutation |
|--|--|
| <code>ValueError: invalid literal for int() with base 10: 'x'</code> | Replace last instance of 'x' with a random integer between -1 and 10 |
| <code>ValueError: could not convert string to float: 'x'</code> | Replace last instance of 'x' with random float between -1 and 10 |
| <code>ValueError: too many / not enough values to unpack</code> | Append duplicate of / delete last token |
| <code>EOFError: EOF when reading a line</code> | Append duplicate or randomized new token |



Bonus Slide: Additional Mutations for Python: *R*

| Mutation | Description of Input Mutation |
|----------------------|---|
| Insert a token | Inserts new token at a random location |
| Split delimited list | Splits one line of input into many using a delimiter, often white space |
| Swap a token | Modify one of the input tokens |
| Remove a token | Remove a random token from the input |
| Empty the input | Replaces entire input with an empty sequence |



RQ3 Bonus: Are InFix's Design Assumptions Valid?

The Design Assumptions

Are they Valid?

1. **Error message templates** are critical for InFix?



Yes! Mutation-only version of InFix solves just **64.5%**

2. Additional **random mutations** are critical for InFix?



Yes! Error template only version of InFix solves just **45.2%**

3. Is our **algorithm structure** critical for InFix?



Yes! Non-hierarchical version solves 96.5%, but the average **number of probes is 4.10** compared to 2.98

RQ3 Bonus: Are InFix's Design Assumptions Valid?

- Two main error-message types: **syntactic** (33%) and **semantic** (66%)
- The input grammars of novice programs can be **surprisingly complex**
Are some types of errors more common than others?
- The error messages are not uniform: **ValueError** is raised by **54.5%** of input-related errors. The following subtypes account for **51.2%**
 - `ValueError: invalid literal for int`
 - `ValueError: could not convert string to float`
 - `ValueError: not enough/too many values to unpack`



RQ4: How sensitive is InFix to input parameters?



InFix **is insensitive** to resource parameters, and thus usable with tight budgets

InFix repairs a **non-trivial amount** of input-related errors **in a single iteration**

RQ4: How sensitive is InFix to input parameters?

Number of Threads

| | 1 | 2 | 3 | 4 | 5 |
|-----|-------|-------|-------|-------|-------|
| 1 | 30.8% | 36.4% | 39.9% | 42.6% | 44.6% |
| 5 | 64.1% | 72.7% | 77.3% | 80.3% | 82.6% |
| 10 | 73.6% | 81.0% | 84.5% | 86.7% | 88.4% |
| 20 | 80.5% | 86.1% | 88.8% | 90.6% | 91.7% |
| 30 | 83.1% | 88.2% | 90.5% | 92.0% | 93.0% |
| 60 | 86.7% | 91.0% | 92.7% | 93.8% | 94.5% |
| 500 | 92.5% | 94.5% | 95.3% | 95.8% | 96.1% |



RQ4: How sensitive is InFix to input parameters?

Number of Threads

| | 1 | 2 | 3 | 4 | 5 |
|--------------------------|-------|-------|-------|-------|-------|
| Maximum Number of Probes | | | | | |
| 1 | 30.8% | 36.4% | 39.9% | 42.6% | 44.6% |
| 5 | 64.1% | 72.7% | 77.3% | 80.3% | 82.6% |
| 10 | 73.6% | 81.0% | 84.5% | 86.7% | 88.4% |
| 20 | 80.5% | 86.1% | 88.8% | 90.6% | 91.7% |
| 30 | 83.1% | 88.2% | 90.5% | 92.0% | 93.0% |
| 60 | 86.7% | 91.0% | 92.7% | 93.8% | 94.5% |
| 500 | 92.5% | 94.5% | 95.3% | 95.8% | 96.1% |



RQ4: How sensitive is InFix to input parameters?

Number of Threads

| | 1 | 2 | 3 | 4 | 5 | |
|--------------------------|-----|-------|-------|-------|-------|-------|
| Maximum Number of Probes | 1 | 30.8% | 36.4% | 39.9% | 42.6% | 44.6% |
| | 5 | 64.1% | 72.7% | 77.3% | 80.3% | 82.6% |
| | 10 | 73.6% | 81.0% | 84.5% | 86.7% | 88.4% |
| | 20 | 80.5% | 86.1% | 88.8% | 90.6% | 91.7% |
| | 30 | 83.1% | 88.2% | 90.5% | 92.0% | 93.0% |
| | 60 | 86.7% | 91.0% | 92.7% | 93.8% | 94.5% |
| | 500 | 92.5% | 94.5% | 95.3% | 95.8% | 96.1% |



RQ5: How does expertise affect InFix's helpfulness?

Setup

- Collected self-assessment of **relative Python ability** from 97 participants
 - Beginner = <1 semester
 - Intermediate = 1–2 semesters
 - Expert = 3+ semesters
- Classified the 60 stimuli into **three difficulty levels**
 - Three expert annotators
 - Fleiss' Kappa $k = 0.71$

Results

Helpfulness of InFix's repairs depending on experience:

| | # | Participant Experience Level | | |
|--------------------|----|------------------------------|----------|--------|
| | | Minimal | Moderate | Expert |
| Easiest Stimuli | 14 | 4.7 | 4.9 | 5.1 |
| Hardest Stimuli | 11 | 3.6 | 4.8 | 5.1 |
| All Stimuli | 60 | 4.0 | 4.8 | 5.2 |

- After controlling for expertise, relative expertise **does not affect helpfulness**



RQ5: How does expertise affect InFix's helpfulness?



After controlling for program difficulty, InFix's repairs are
equally helpful for novices and relative experts