



**Open Universiteit**

# Grammar-Based Action Selection Rules for Scriptless Testing

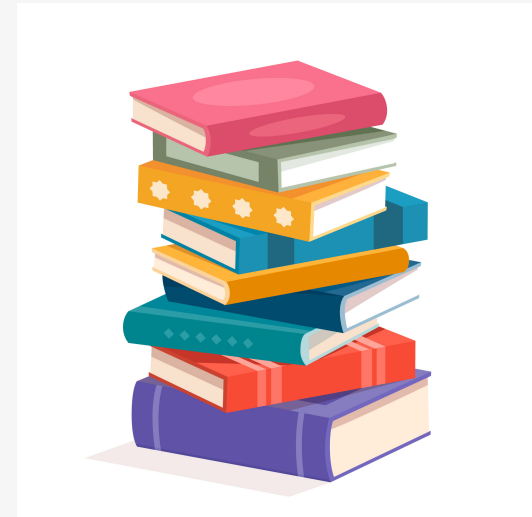
*Lianne V. Hufkens*

13 June 2023



# Grammar-Based Action Selection Rules for Scriptless Testing

- Scriptless GUI Testing
- Grammar-based Action Selection
- Integration
- Experiments
- Results
- Conclusions & Future Work
- Extras



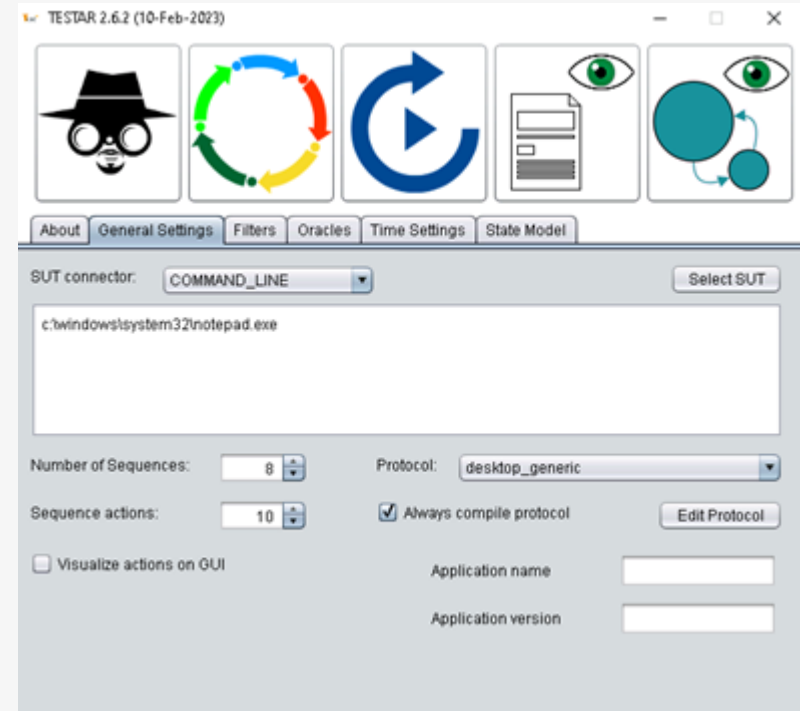
DARE

# Scriptless GUI Testing

# TESTAR tool



- <https://testar.org/>
- [https://github.com/TESTARtool/TESTAR\\_dev/](https://github.com/TESTARtool/TESTAR_dev/)
- Open Universiteit
- Universidad Politècnica de Valencia
- European project
- Academic open source tool for scriptless GUI testing



# The Whys of Automated GUI Testing

Software testing = executing a program or application with the intent of finding failures

## Why test?

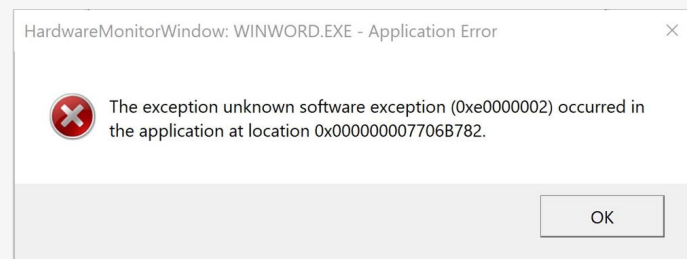
- No software is born faultless
- Late fixes are more expensive (time, effort)

## Why Graphical User Interface?

- Many applications have GUI
- GUI testing has the end user perspective

## Why automated?

- Manual testing takes a lot of time, skill and knowledge
- Automation can handle complex and large software



# Scriptless Testing

Test sequences are generated during execution

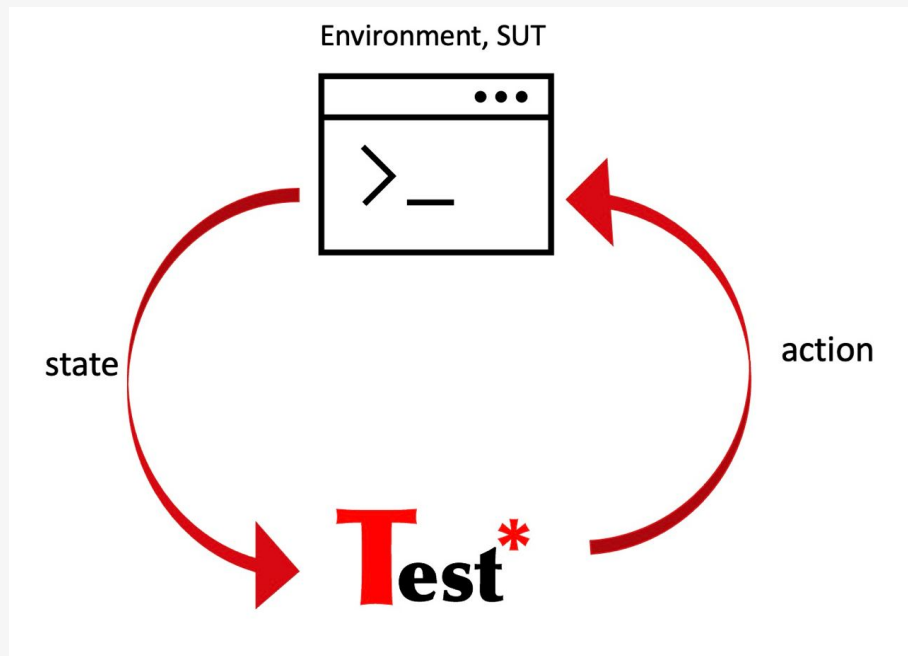
- Steps are based on available actions
- Checks for failure after every action
- Strategies decide which action to select

Pros:

- Low maintenance
- Reliable, works even if the SUT changes (on-the-fly generation)

Cons:

- Less specific results, harder to interpret
- No concrete use cases or work-flows



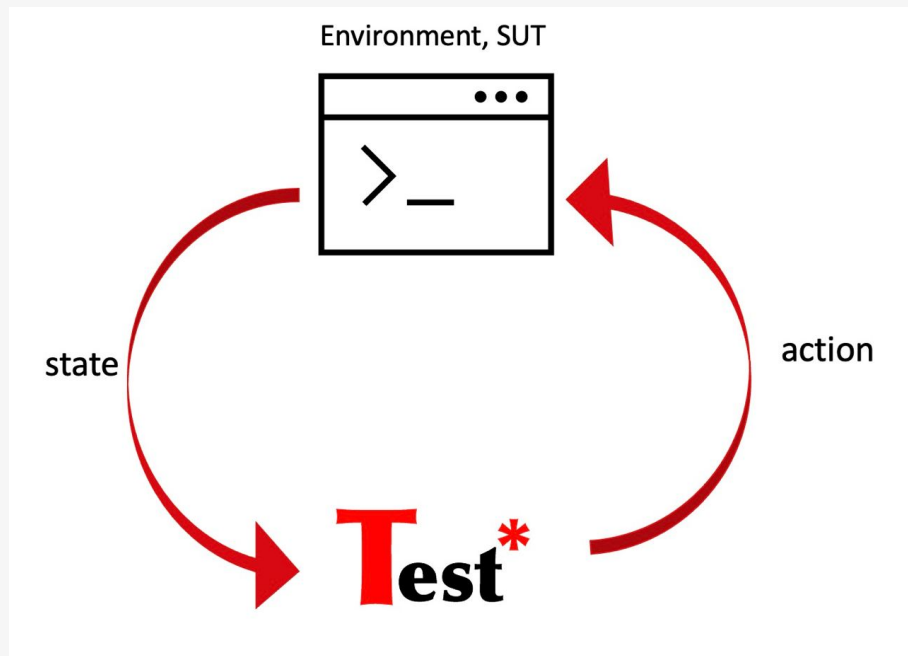


# Scriptless Testing: A Process of Action Selection

- Action Selection:
  - A constant question of what action to pick

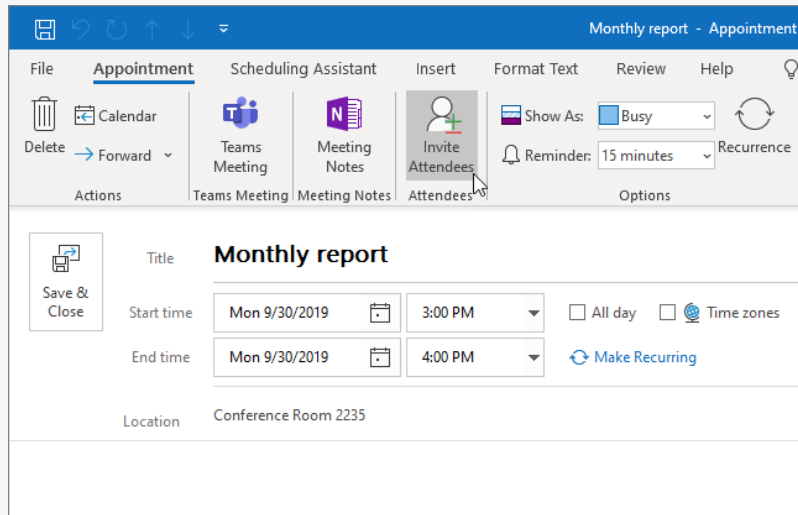
## Considerations:

- Dynamic environment
  - Available actions may change
- Prioritization – yes or no?
  - What logic to apply?
- Destination unclear
  - Where are the bugs?



# Action Selection and Randomness

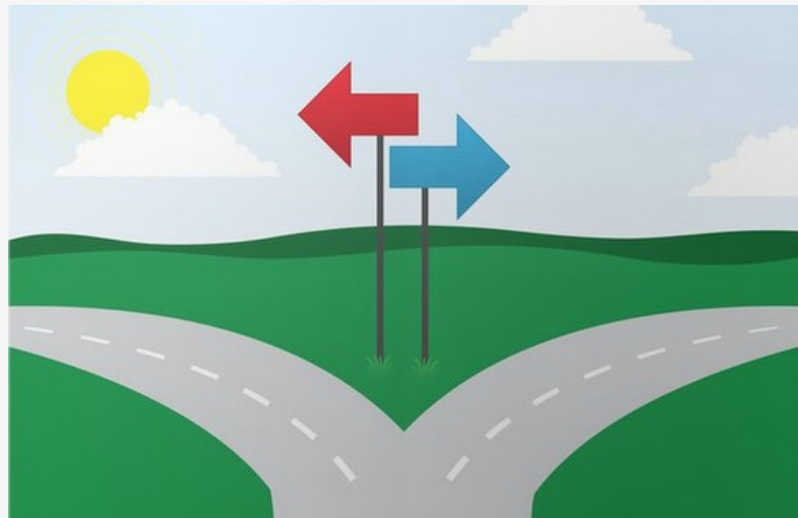
- Existing work employs stochastic methods
  - Aka random testing
- Good for robustness, exploration, can deal well with dynamic environments
- But...
  - No direction, not human-like
  - Fails at tasks that require successive steps





# How to Test Smarter?

- Preferably:
  - More direction
  - Integrate more human-like behavior
  - Keep the benefits of scriptless testing:
    - Exploration
    - Robustness
    - Adaptability to dynamic environments
    - Low effort to create and maintain



**D  
A  
R  
E**

# Grammar-based Action Selection



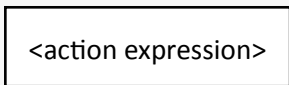
# Context-Free Grammar for Test Strategies

```
(strategy) ::= (action_list) | (if_else)
(action_list) ::= (action)+
(if_else) ::= if (bool_expr) then (action_expr)
            else (action_expr)
(bool_expr) ::= not (bool_expr) | (bool_expr)
              (bool_opr) (bool_expr) | (int_expr)
              (int_opr) (int_expr) | (state_bool)
              | [bool]
(action_expr) ::= (if_else) | (action_list)
(bool_opr) ::= and | or | xor
(int_opr) ::= > | >= | < | <= | == | !=
(int_expr) ::= (n_actions) | [int]
(state_bool) ::= state-changed | any-exist
              (modifier)? (related_action) | any-exist
              (modifier)? (action_type)? | sut
              (filter) (sut_type)
(n_actions) ::= n-actions (visit_modifier)?
              (action_type)?
(action) ::= [int]? select-previous |
            [int]? select-random (modifier)?
            (action_filter)? | [int]? select-random
            (visit_modifier)? (related_action)
(filter) ::= of-type | not-of-type
(action_type) ::= click-action | type-action |
               drag-action | hit-key-action
               | input-action | form-field-action
               | form-submit-action
(modifier) ::= most-visited | least-visited
            | visited | unvisited
(related_action) ::= sibling-action | child-action
                 | sibling-or-child-action
(sut_type) ::= windows | linux | android | web
```

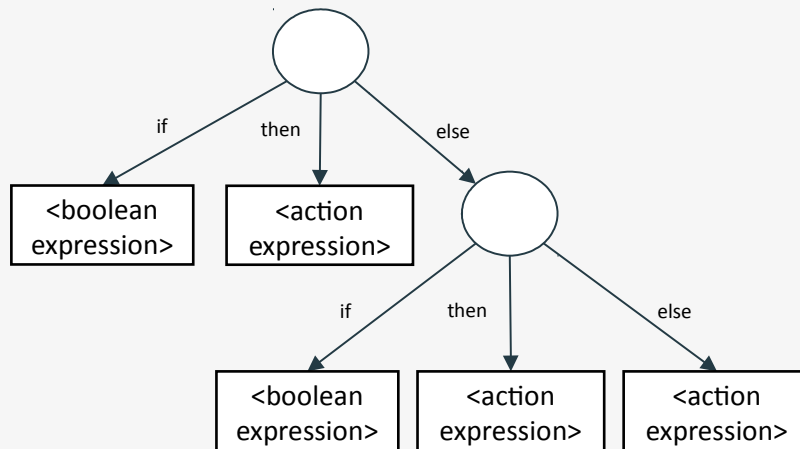
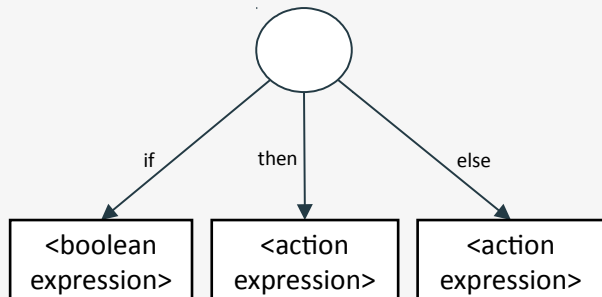
- Grammar to write strategies
- Strategy = Action Selection Rules (ASR)
  - How to select the next action
- Allows dynamic action prioritization
- Enables human-like approaches
- Readable for humans
- ASR can be generated (computer) or hand-crafted (human)

# How to Write an ASR – Structure

- <action expression> = a weighted list of <action>
  - Can also be just one <action>
- <action> = an instruction that can be executed
- <boolean expression> = condition



or



# Basic Syntax

<boolean  
expression>

<action>

any-exist

select-previous

sut

select-random

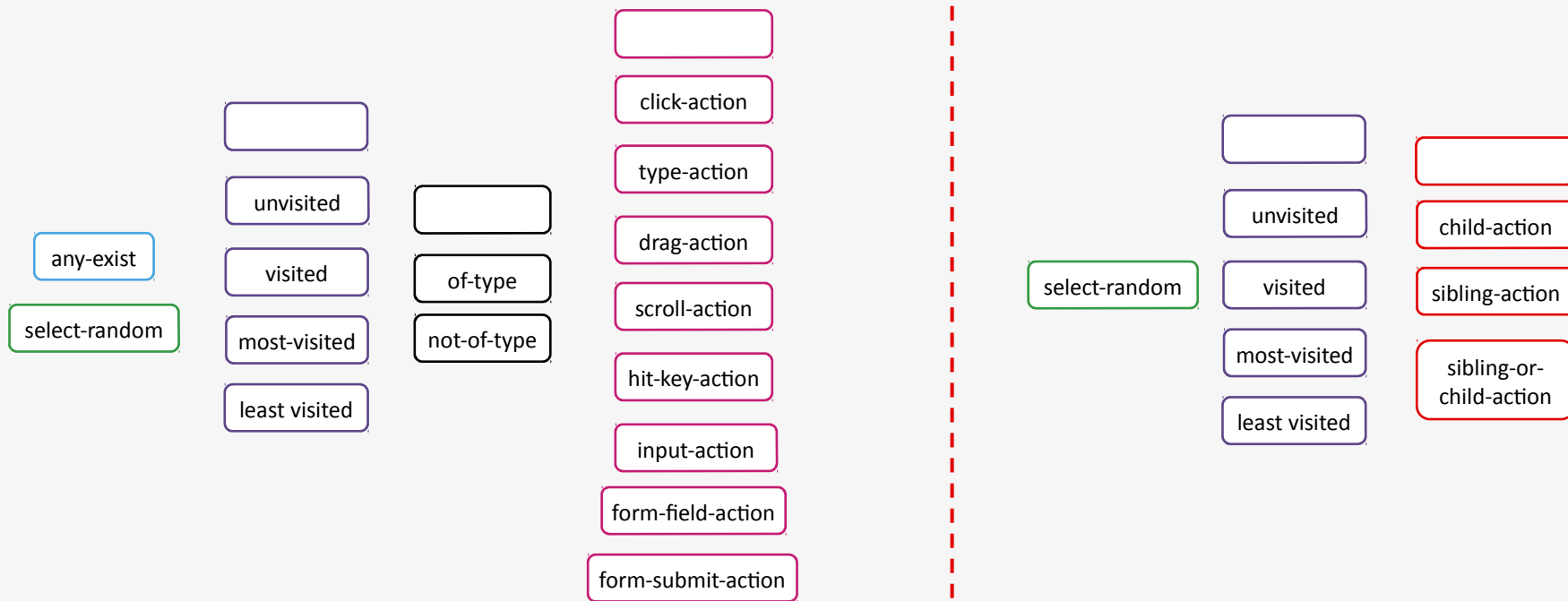
state-changed

n-actions

- An <action> is executed as is
  - Can be made more specific
- <boolean expression> checks some aspect of the current state
  - If there are actions available, System Under Test, if the last action changed the state, how many actions are available...
- There will be more options in the future

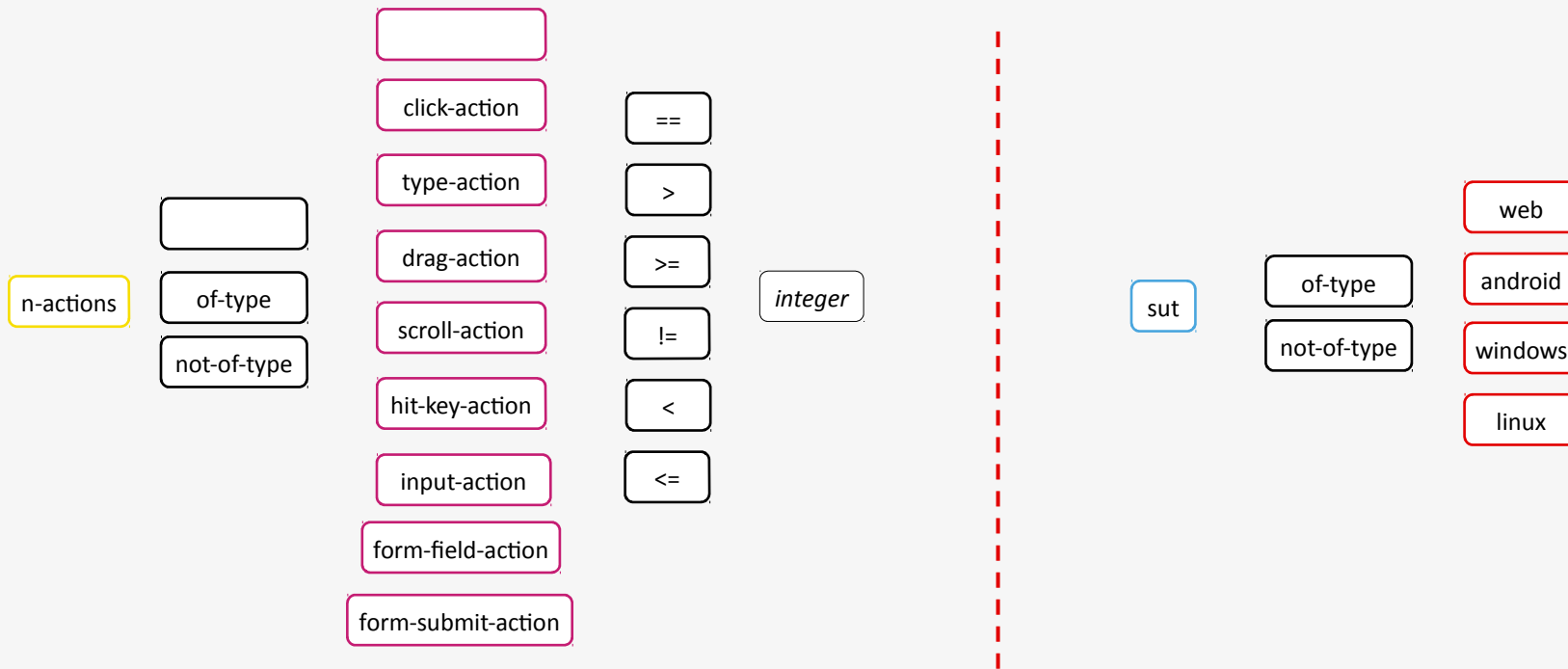


# Basic Syntax

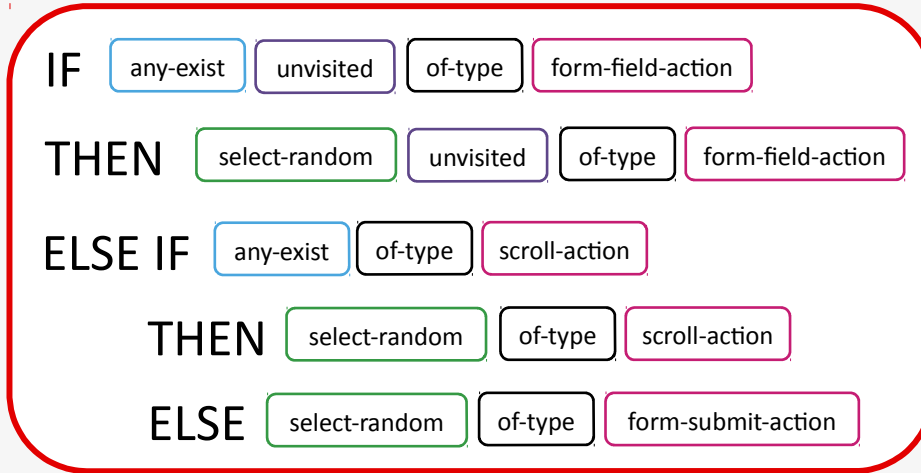




# Basic Syntax



# How to Write an ASR





**DARE**

## Integration



# How to Make the Grammar Work

- So now there is a grammar! It is ready to go, right?
  - Not quite.
- Associate semantic expressions with corresponding widgets and actions
  - Ex: <click-action> corresponds to left clicks and right clicks, <drag-action> to dragging actions
- Provide input strings for text fields
  - Detect text field types and match string pattern
  - Email, urls, dates, times, integers...



# Abstraction

- Abstraction is needed to identify states, widgets, and derive actions
- Track widgets and actions
  - Ex: Is an action unvisited or not?
- A few more questions:
  - When is the state considered changed?
  - Is this one widget the same as the previous?
  - If the input is different, is it still the same action?
- In short: abstraction level matters!

# Now What? Time to Test

- With grammar one can write ASRs
  - Do they work?
  - Are they reliable?
  - Can we create something better than random?
- So... let's find out!



**DARE**

## Experiments: Form-filling

# The Challenge of Form-filling

- Web forms are found in many places
  - Login screens, questionnaires, email fields, filtering options, etc.
- Scriptless testing struggles here
  - Many fields to interact with
  - One bad action can undo all progress
  - There may be an order to follow
  - Input formats can be restricted

The form contains the following elements:

- Name \***: Two input fields, one labeled "First" and one labeled "Last".
- Email \***: A single wide input field.
- Phone**: An input field with a small square icon and a dropdown arrow on the left.
- Submit**: A button at the bottom of the form.



# Human-based Form-filling ASR

IF any-exist unvisited of-type form-field-action  
THEN select-random unvisited of-type form-field-action  
ELSE IF any-exist of-type scroll-action  
THEN select-random of-type scroll-action  
ELSE select-random of-type form-submit-action

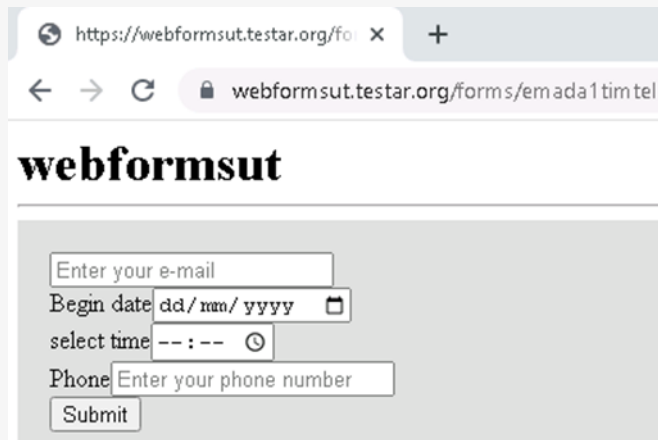
- `<form-field-action>` are the children of a form (except submit)
- `<form-submit-action>` refers to submit actions
- Prioritizes getting input into all the fields
  - Scroll if needed
  - Hit submit at the end

# Experiment with Generated Forms

- WebformSUT generates arbitrary web forms
  - <https://github.com/TESTARtool/webformsut>

- From input fields as specified by W3C:

- |                  |          |
|------------------|----------|
| • Button         | • Range  |
| • Checkbox       | • Reset  |
| • Color          | • Search |
| • Date           | • Submit |
| • Datetime-local | • Tel    |
| • Email          | • Text   |
| • Month          | • Time   |
| • Number         | • Url    |
| • Password       | • Week   |
| • Radio          |          |



- (Experiment excludes reset and color fields)



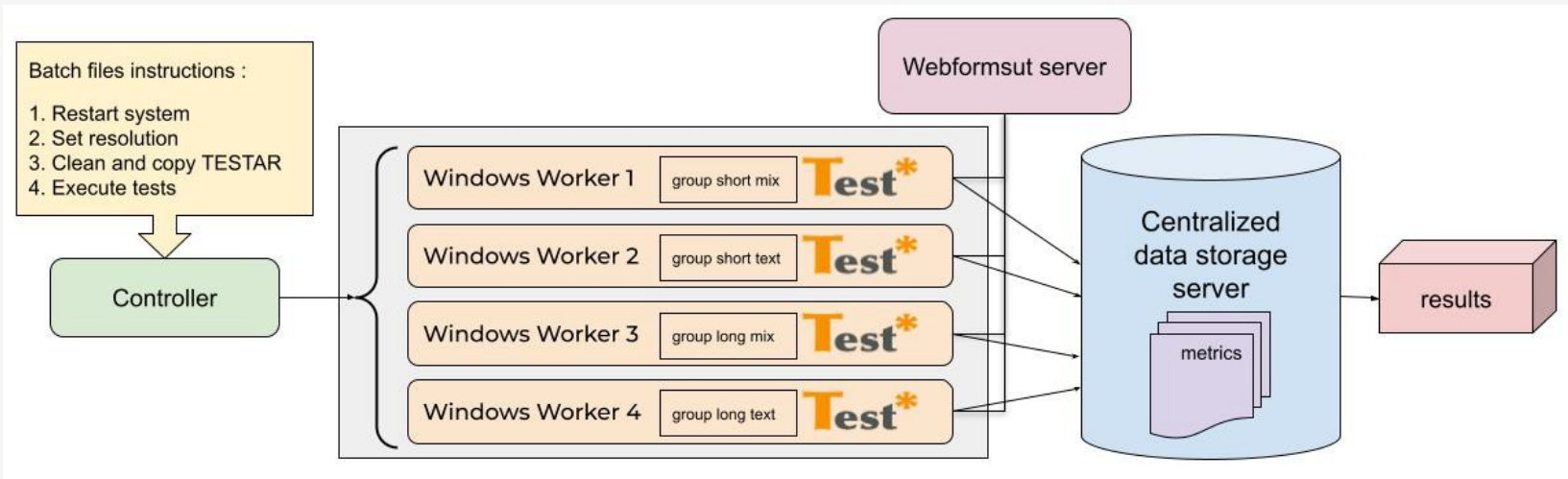
# Experiment with Generated Forms

- Generate four groups of forms:
  - Short (10-30 fields) or long (50-70 fields)
  - Text-only or mixed
- Each group contains 30 unique forms
- Run every form 30 times each for both random and human-based ASR
- Limit actions to 45 (short) or 105 (long)
- Main points of interest:
  - Successful submits
  - Field interactions

The screenshot shows a web browser window with the URL `https://webformsut.testar.org/forms/emada1timtel`. The page title is "webformsut". The form contains the following fields and controls:

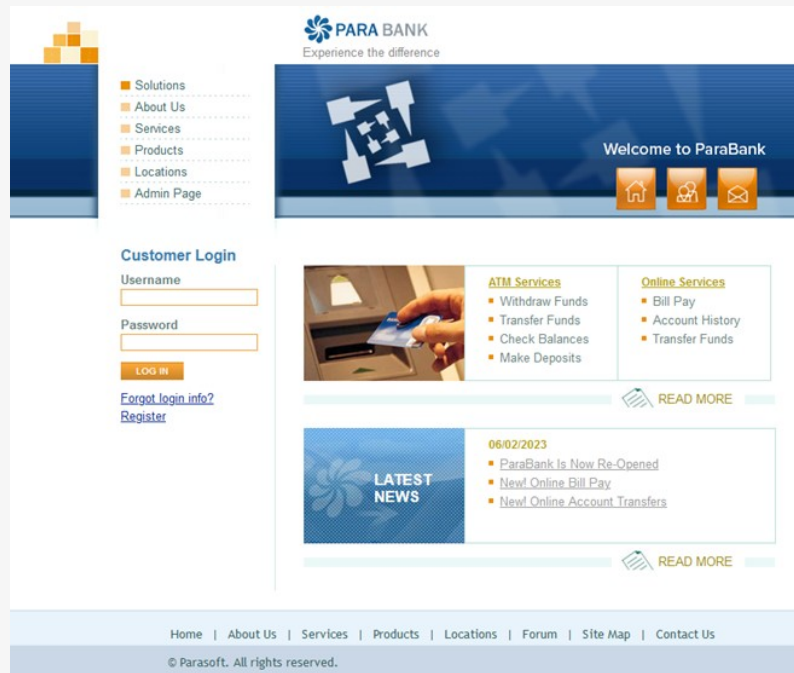
- Text input: "Enter your e-mail"
- Date input: "Begin date" with a dropdown menu showing "dd/mm/yyyy" and a calendar icon.
- Time input: "select time" with a dropdown menu showing "--:--" and a clock icon.
- Text input: "Phone" with a placeholder "Enter your phone number".
- Submit button: "Submit".

# Experiment with Generated Forms



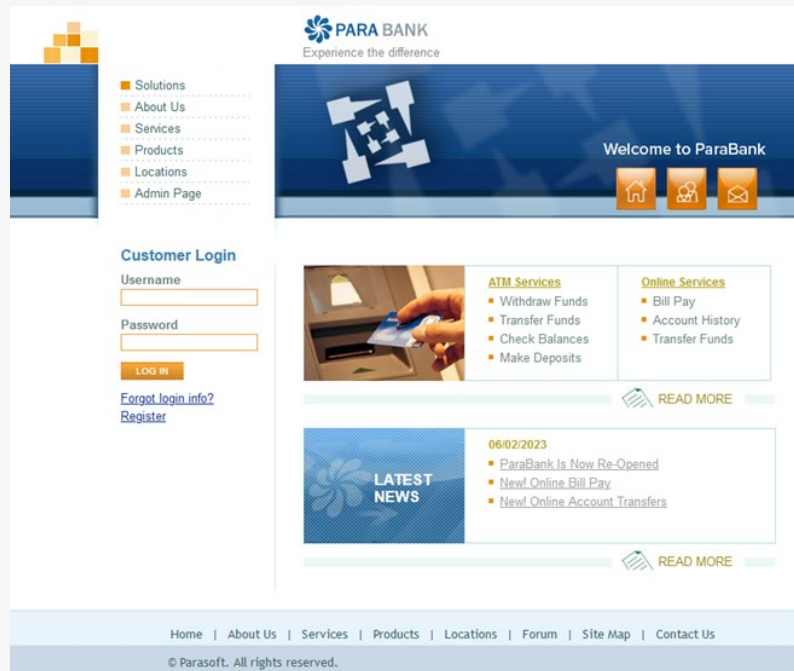
# Experiment with Parabank

- Demo environment
  - Mimics a banking application
  - Variety of field behavior:
    - Prepopulated
    - Mandatory (cannot submit until correct)
    - Required (error if empty)
- 7 web forms in total
  - Open account, update profile, transfer, customer care, bill payment, find transactions, request loan
- Exclude logout and admin buttons

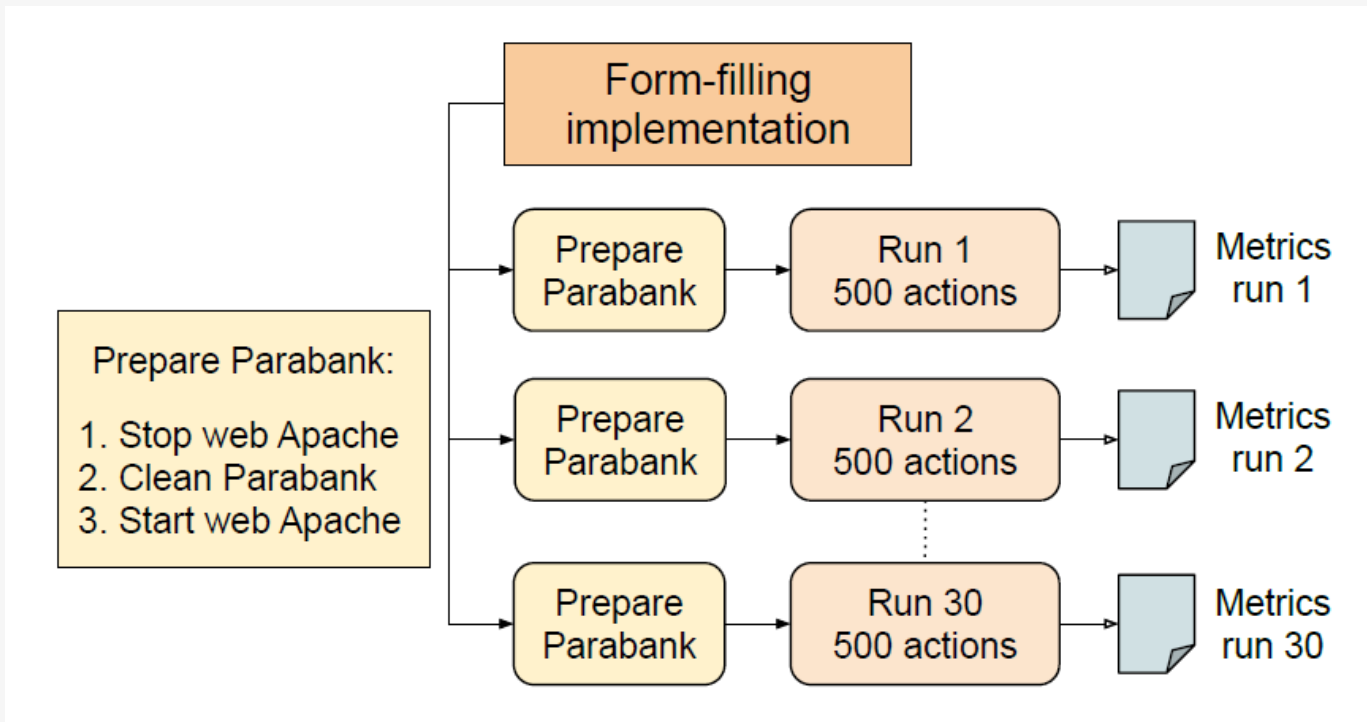


# Experiment with Parabank

- Run a long sequence of 500 actions
  - Repeat 30 times for random and human-based ASR
- Automatically log in at the start
- Allow random exploration
  - Switch to ASR when a form is found
  - Go back to random afterwards
- Main points of interest:
  - Successful and unsuccessful submits
  - Field interactions



# Experiment with Parabank

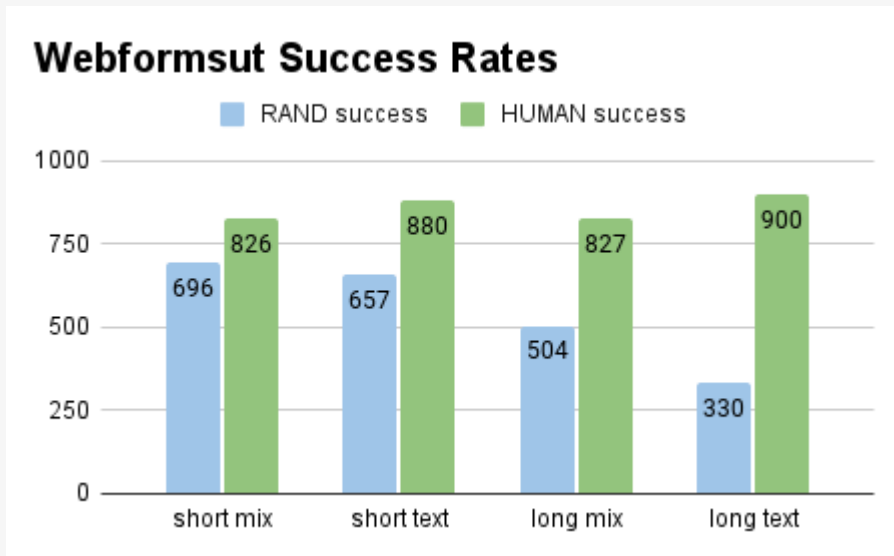


# DARE

## Results

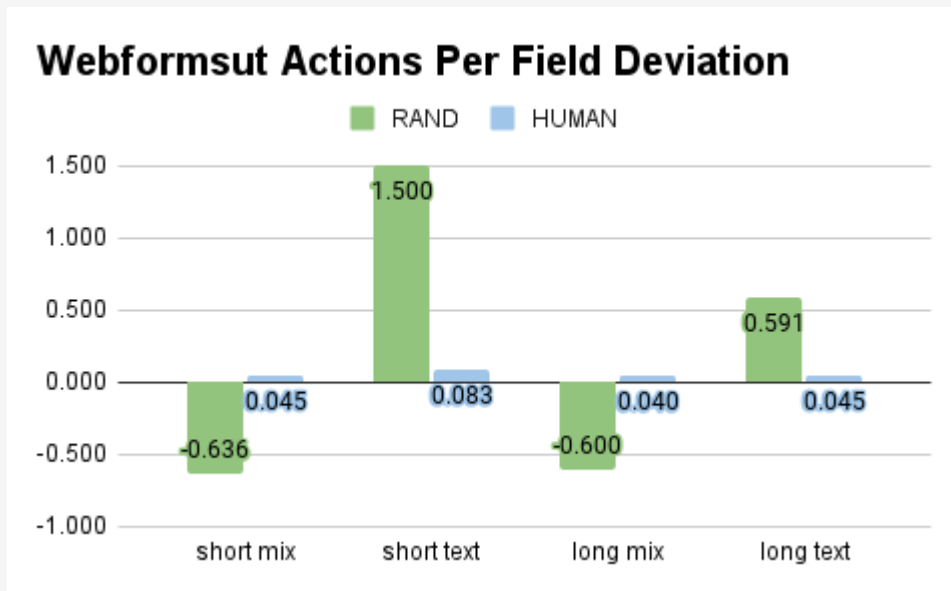


# Generated Forms – Success Rates





# Generated Forms – Actions Per Field







# Parabank – Customer Care Form

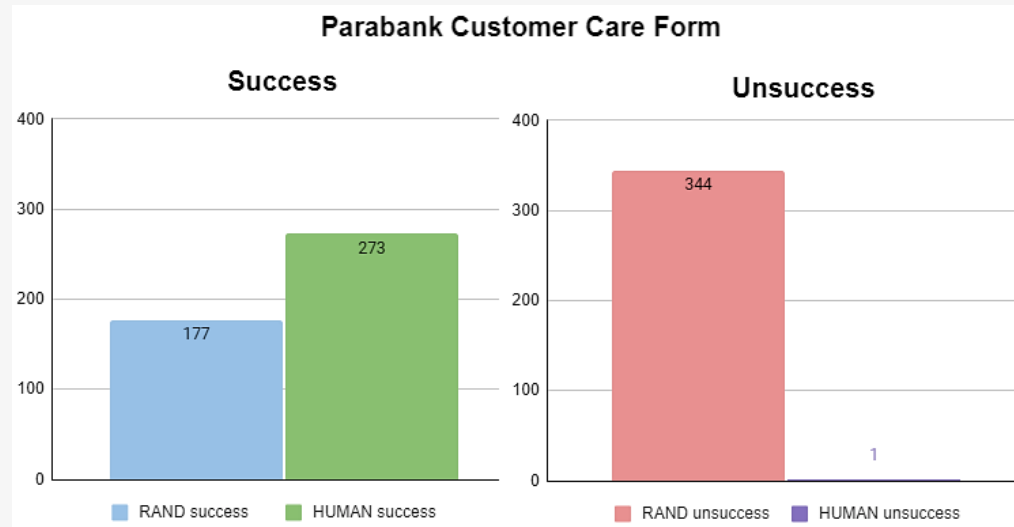
**Customer Care**

Email support is available by filling out the following form.

Name:	<input type="text"/>	Name is required.
Email:	<input type="text"/>	Email is required.
Phone:	<input type="text"/>	Phone is required.
Message:	<input type="text"/>	Message is required.

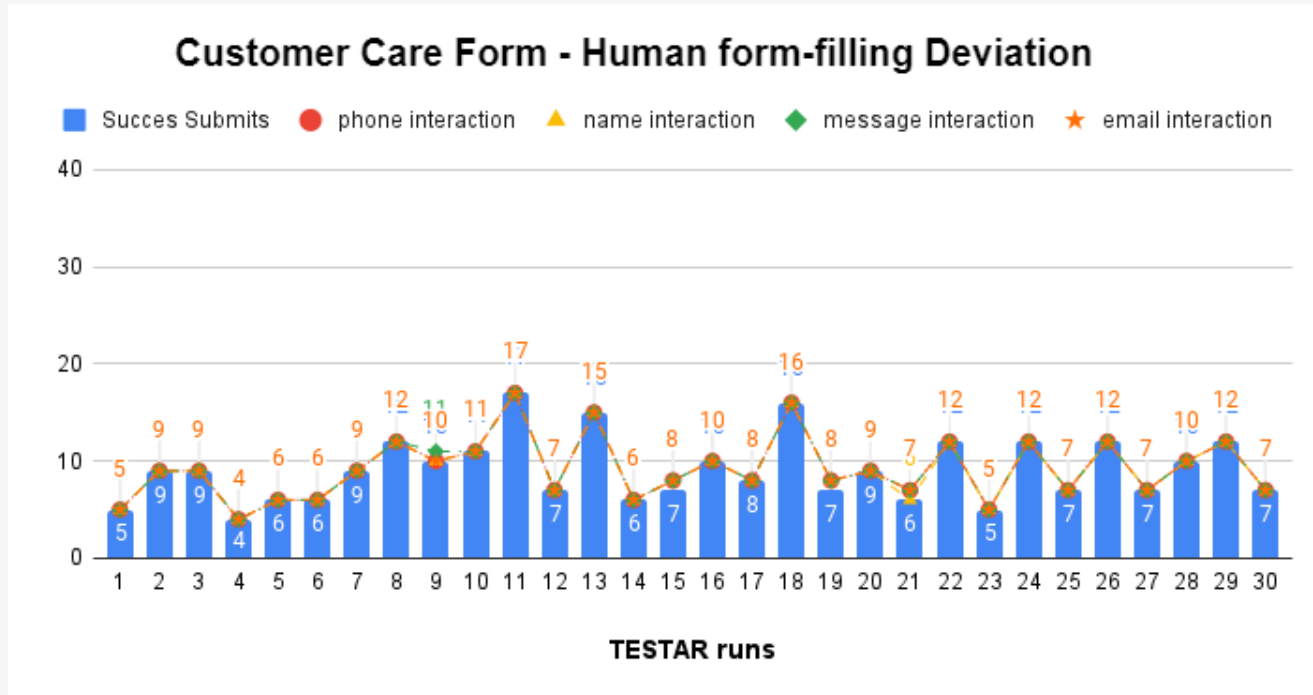
**SEND TO CUSTOMER CARE**

# Parabank – Customer Care Form Success and Unsuccess



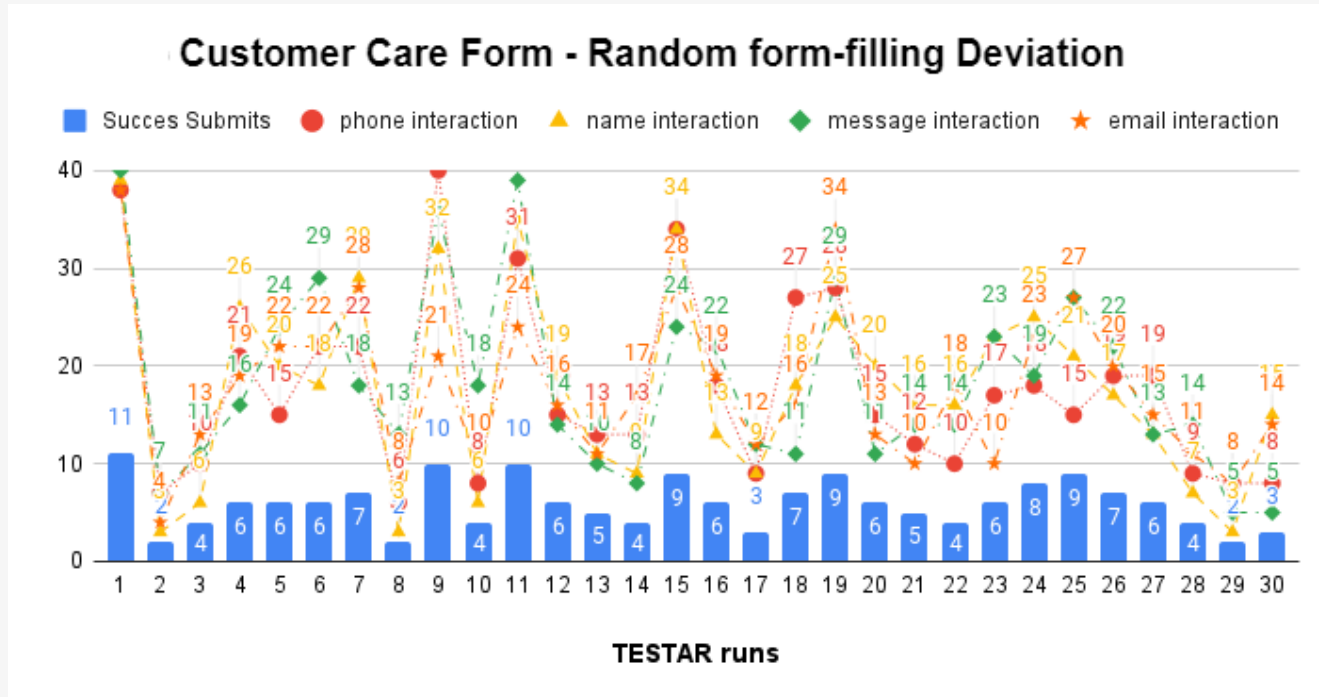


# Parabank – Customer Care Form Interactions





# Parabank – Customer Care Form Interactions



D  
A  
R  
E

## Conclusions & Future Work

# Conclusions

- Human-based ASR outperforms random on form-filling
- Successfully demonstrated grammar-based ASR
  - Maintains scriptless testing benefits (adaptability, robustness, exploration, easy to maintain)
  - Added direction, human-like behaviour
  - Not as rigid or explicit as scripted tests, not as scattered or chaotic as traditional scriptless testing
- New application for grammar-based testing
  - Traditionally used to generate input data





# Future work

- Generate grammar-based ASRs
  - Evolve ASRs through evolutionary algorithms
  - Compare successful hand-crafted and generated ASRs
- Apply grammar-based ASRs to other SUTs
  - Mobile, desktop, XR applications
- Use grammar to define oracles (mechanisms that determine if tests pass or fail)
  - Look into adding domain, business, user knowledge





# DARE

## Extras



# Parabank – New Account Form

**Open New Account**

What type of Account would you like to open?

CHECKING ▾

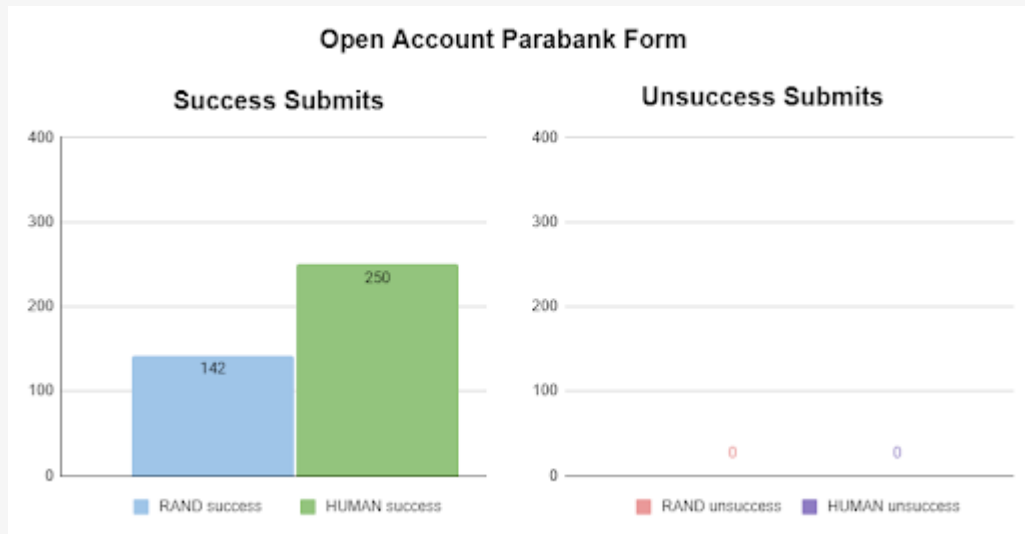
A minimum of 100,00 \$ must be deposited into this account at time of opening.  
Please choose an existing account to transfer funds into the new account.

12345 ▾

**OPEN NEW ACCOUNT**

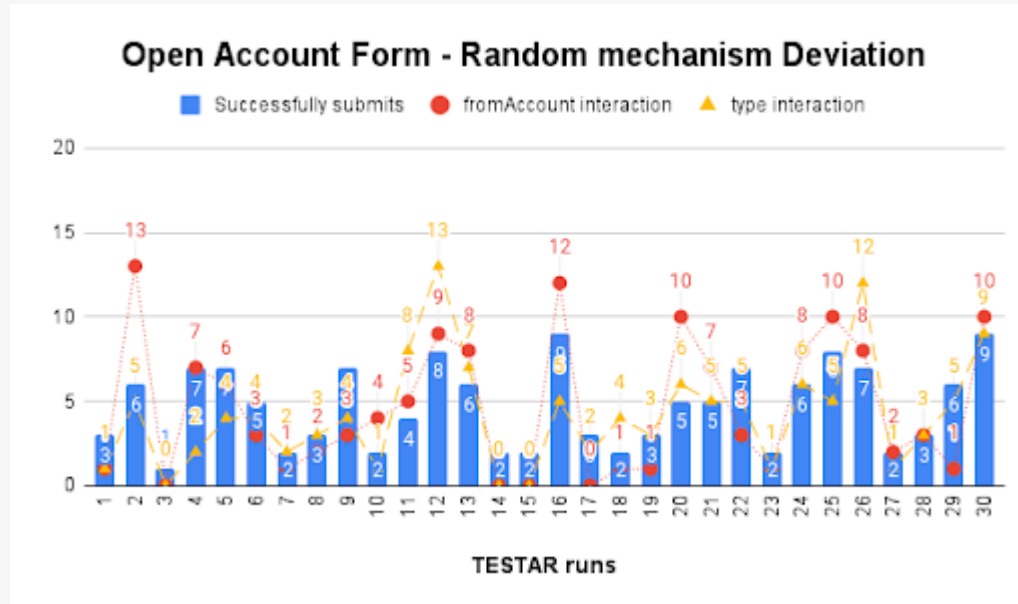


# Parabank – New Account Success and Unsuccess



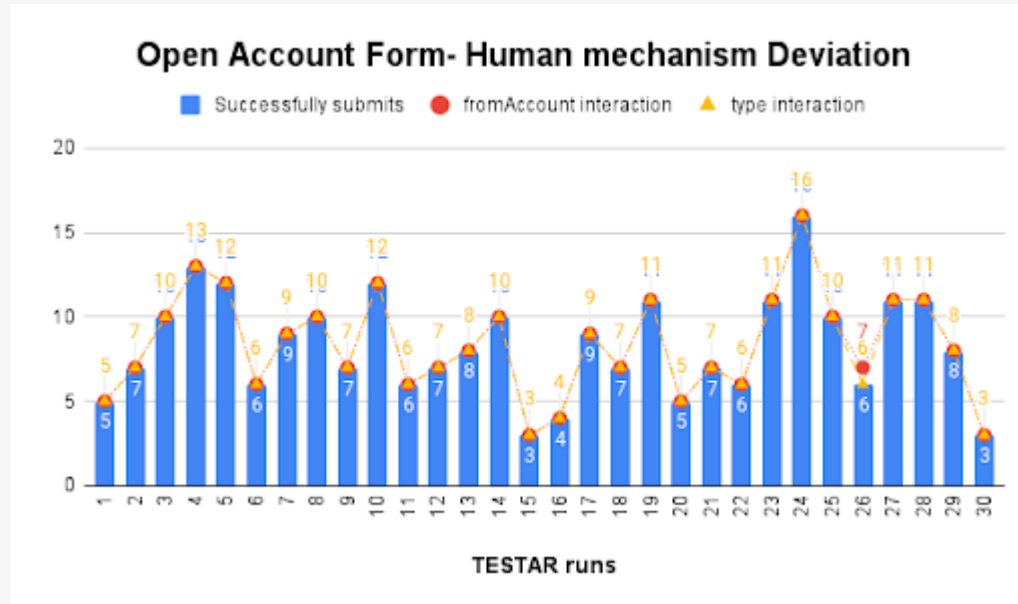


# Parabank – New Account Interactions





# Parabank – New Account Interactions





# Parabank – Update Profile Form

**Update Profile**

First Name:

Last Name:

Address:

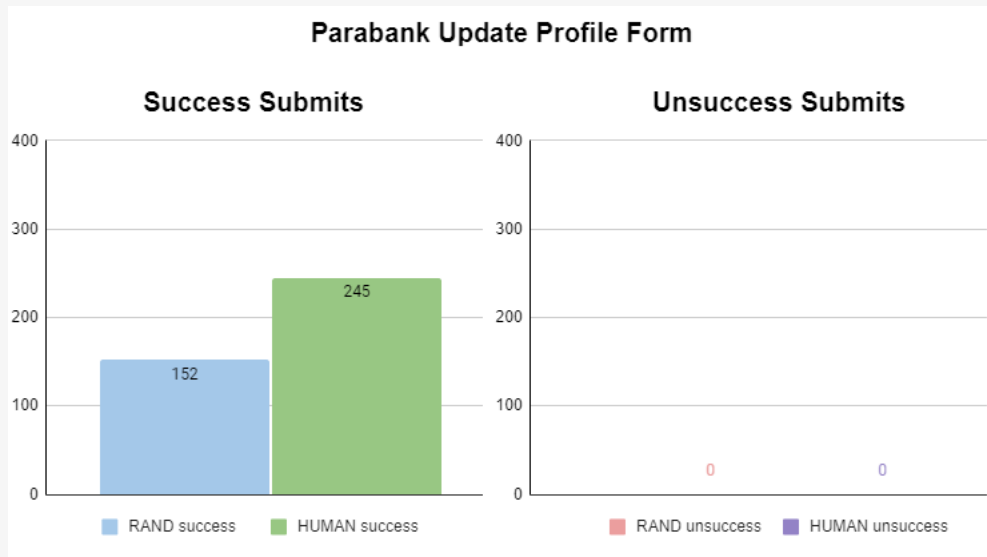
City:

State:

Zip Code:

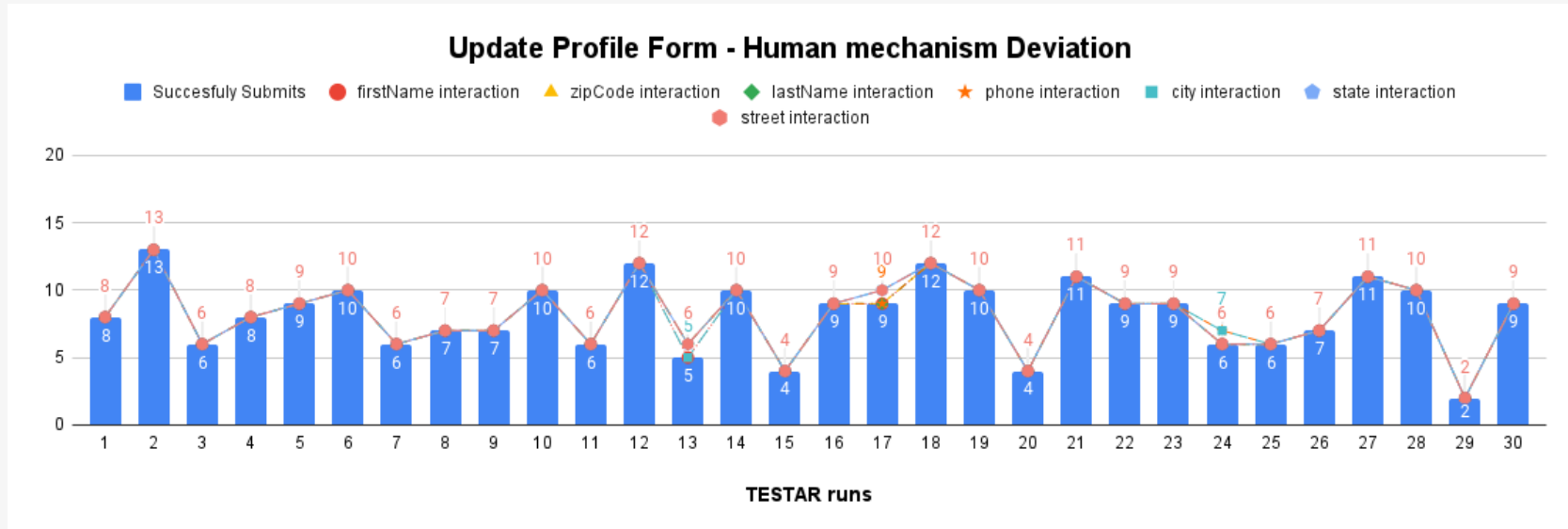
Phone #:

# Parabank – Update Profile Form Success and Unsuccess



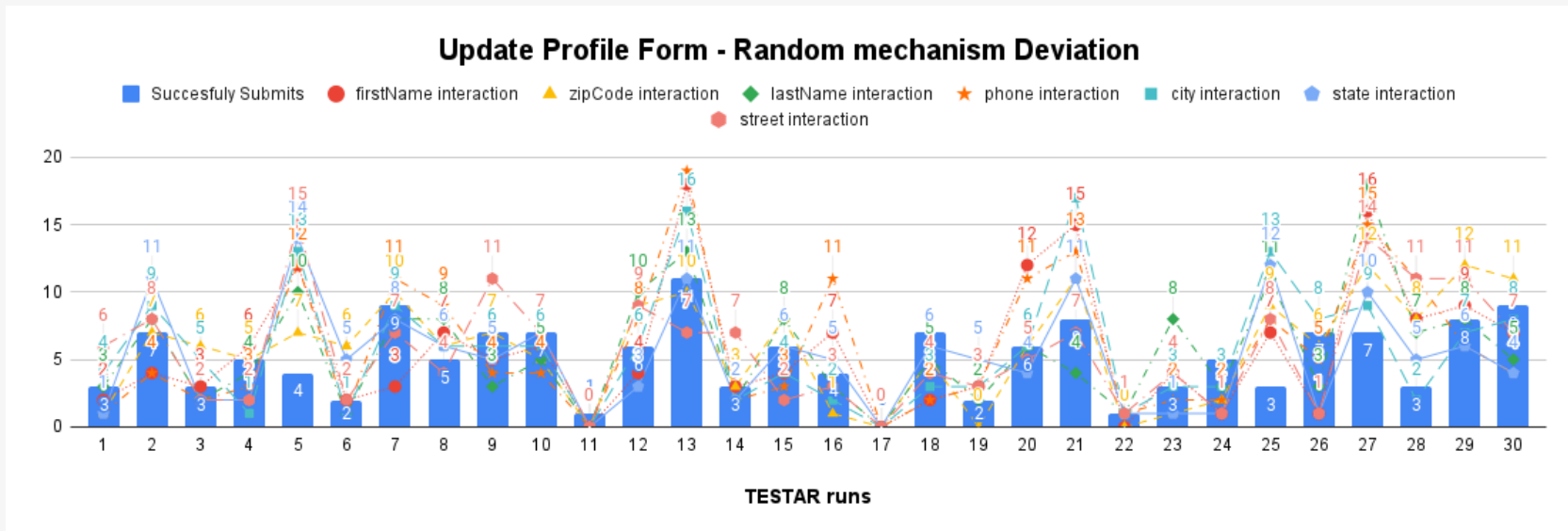


# Parabank – Update Profile Form Interactions





# Parabank – Update Profile Form Interactions





# Parabank – Transfer Funds Form

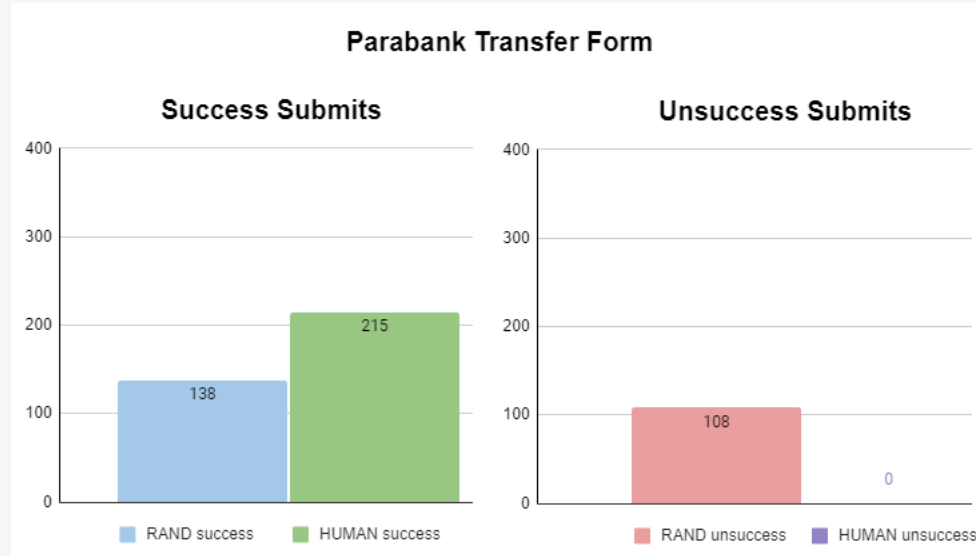
**Transfer Funds**

Amount: \$

From account #  to account #

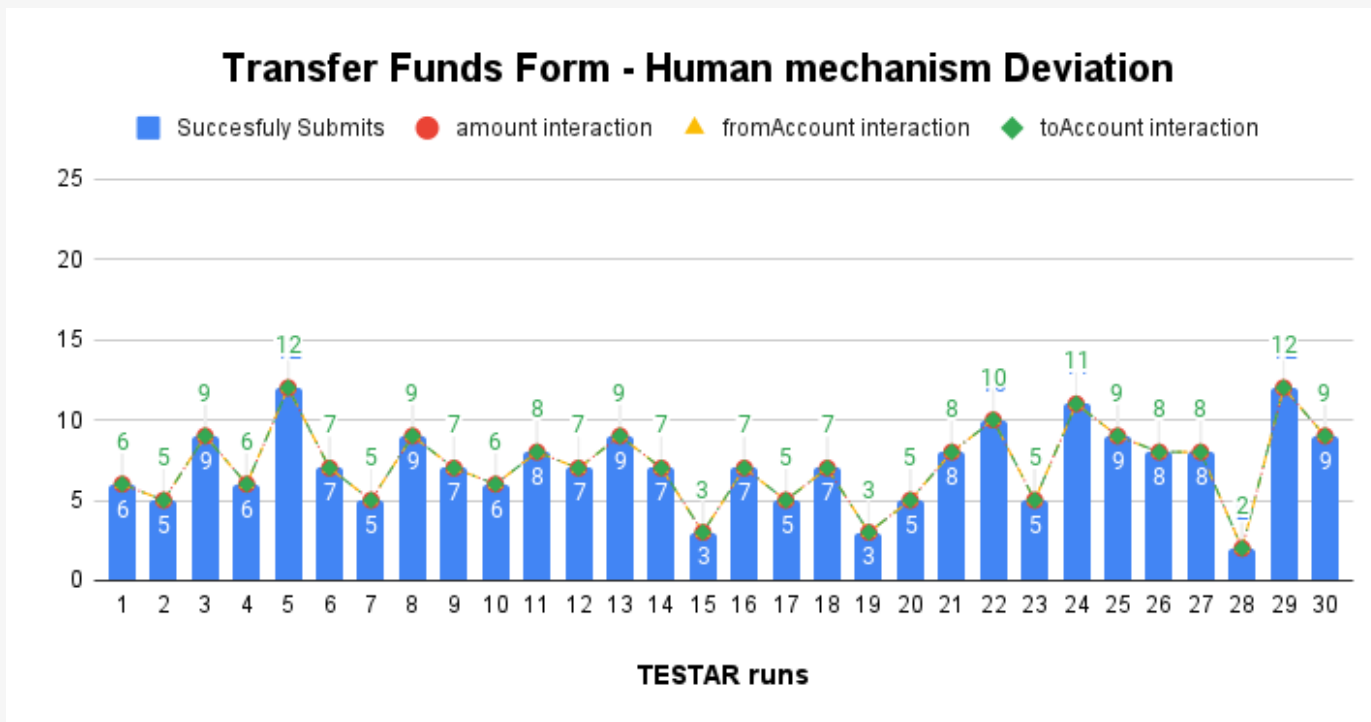
**TRANSFER**

# Parabank – Transfer Funds Form Success and Unsuccess



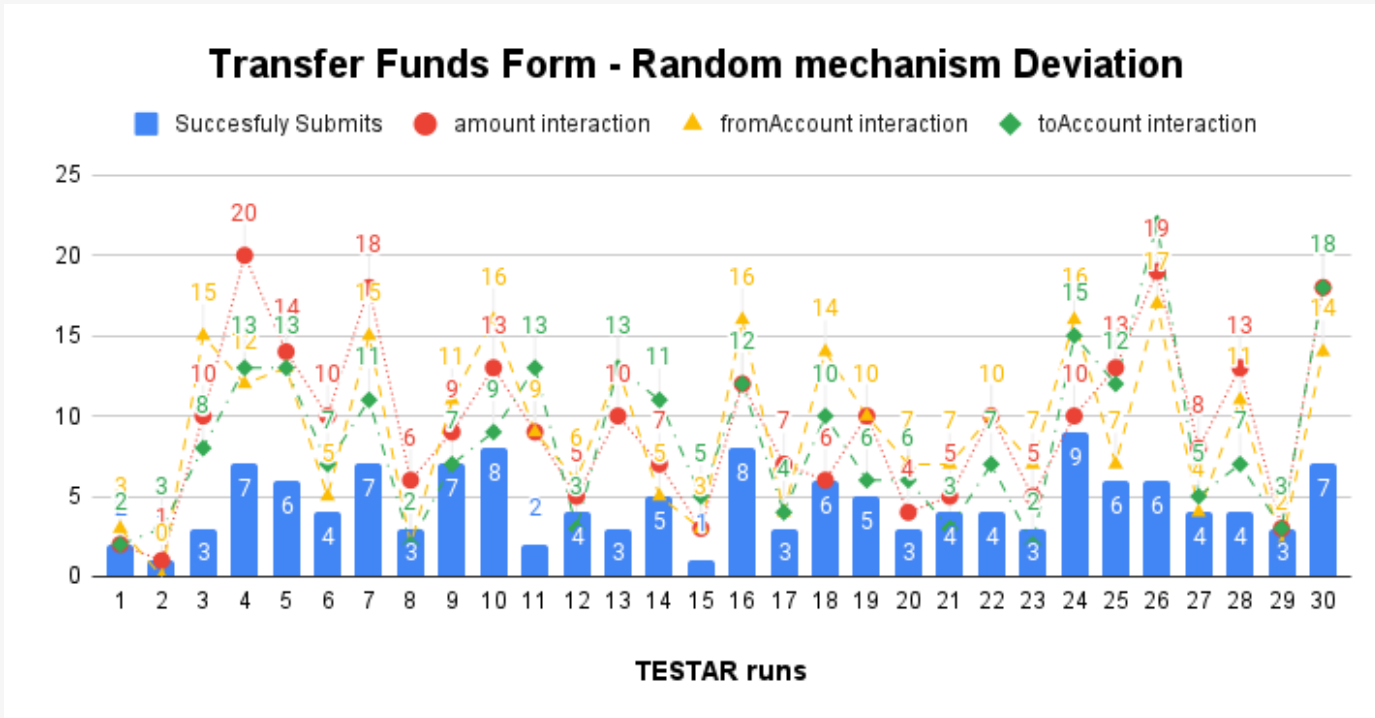


# Parabank – Transfer Funds Form Interactions





# Parabank – Transfer Funds Form Interactions





# Parabank – Bill Payment Form

**Bill Payment Service**

Enter payee information

Payee Name:

Address:

City:

State:

Zip Code:

Phone #:

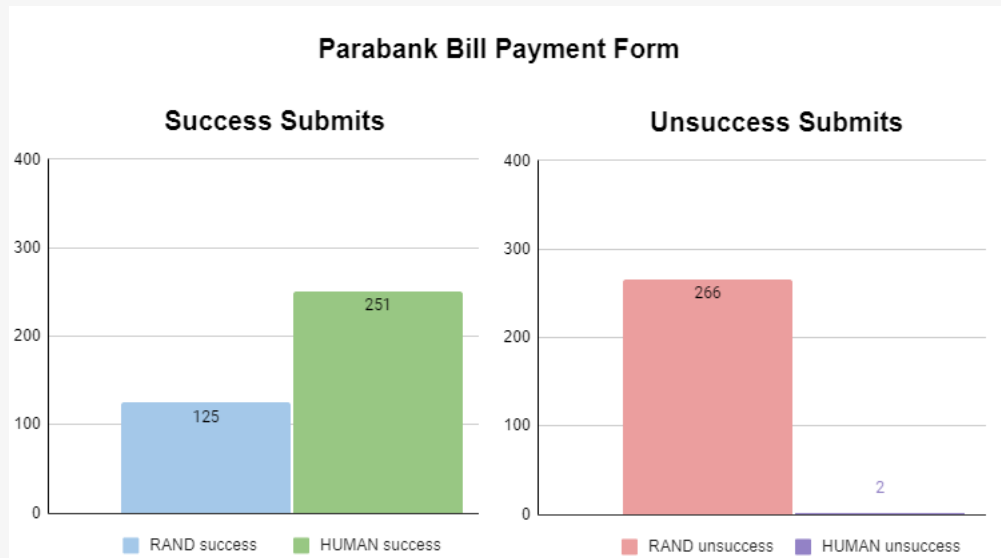
Account #:

Verify Account #:

Amount: \$

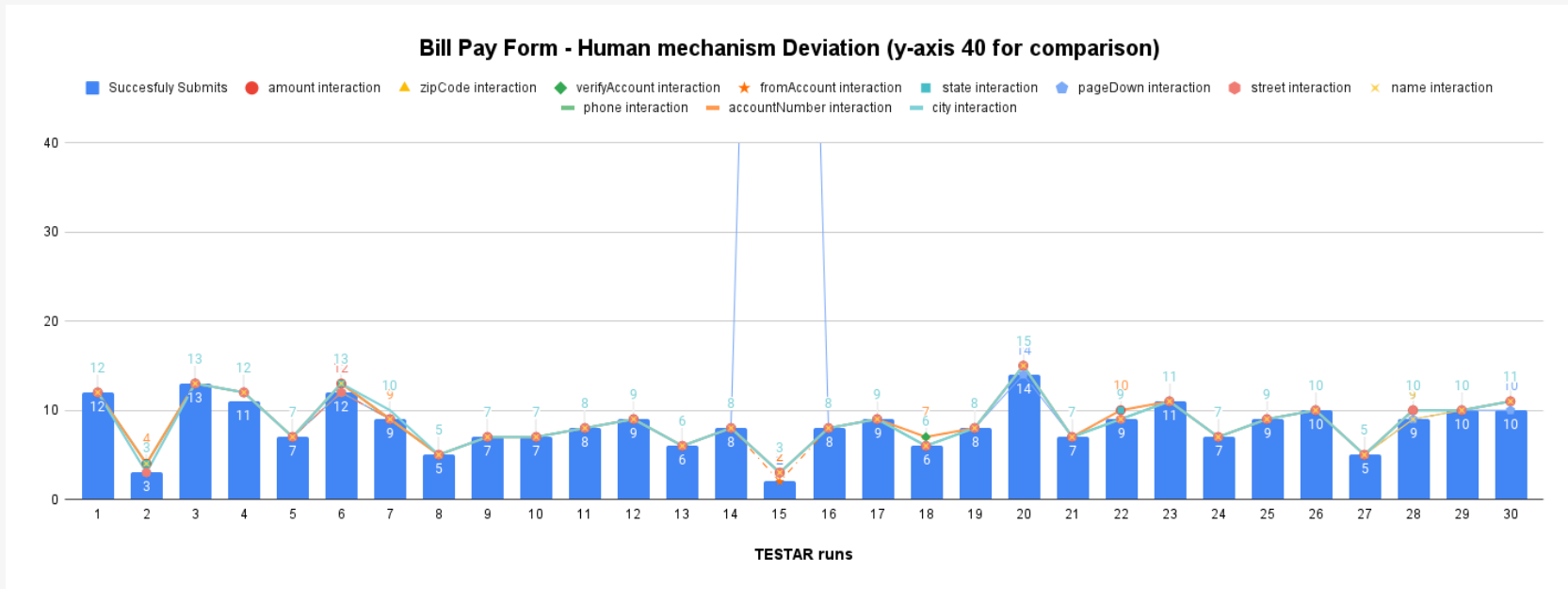
From account #:

# Parabank – Bill Payment Form Success and Unsuccess



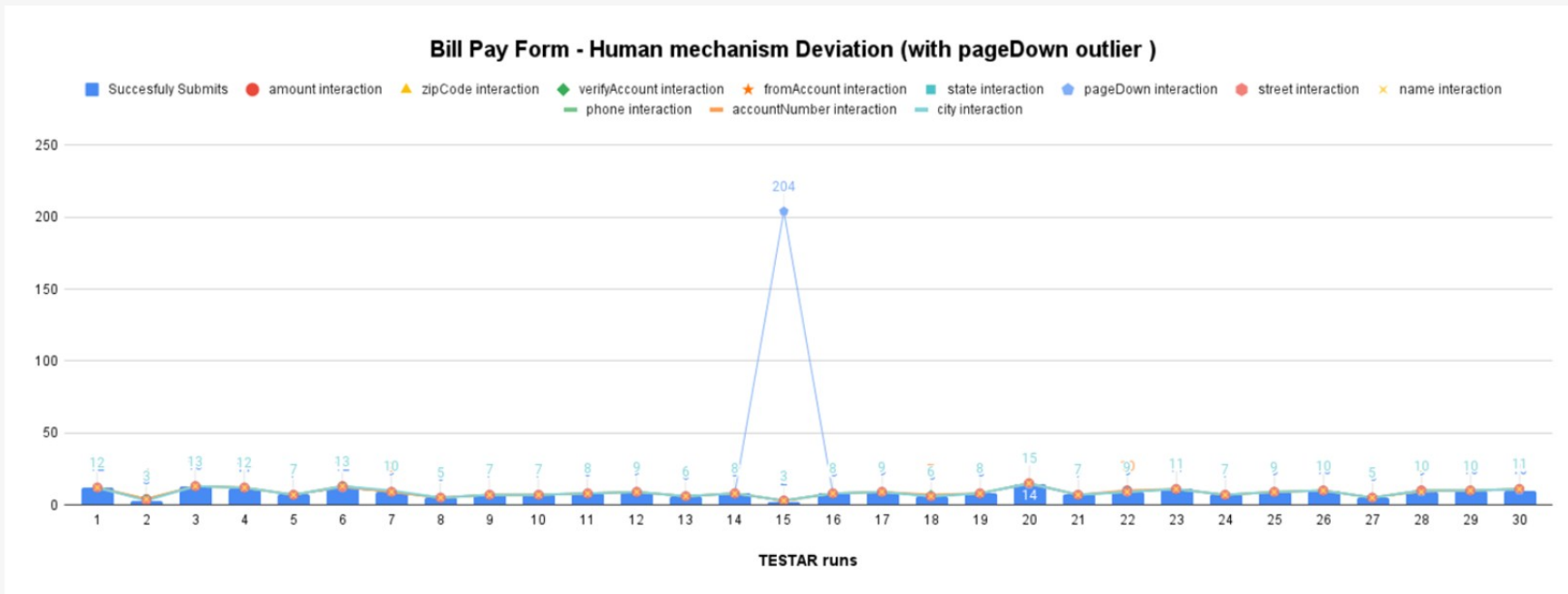


# Parabank – Bill Payment Form Interactions



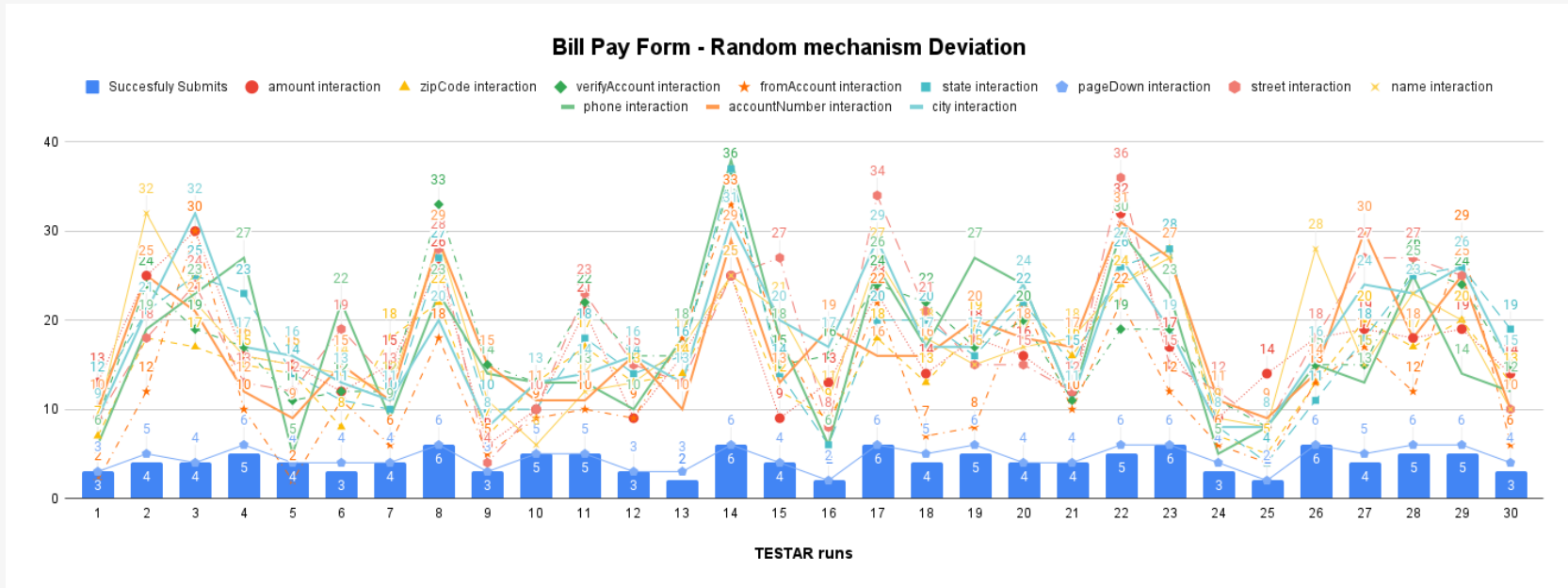


# Parabank – Bill Payment Form Interactions – with outlier






# Parabank – Bill Payment Form Interactions





# Parabank – Find Transactions Form

**Find Transactions**

Select an account:  

---

Find by Transaction ID:

**FIND TRANSACTIONS**

---

Find by Date:  (MM-DD-YYYY)

**FIND TRANSACTIONS**

---

Find by Date Range

Between  and  (MM-DD-YYYY)

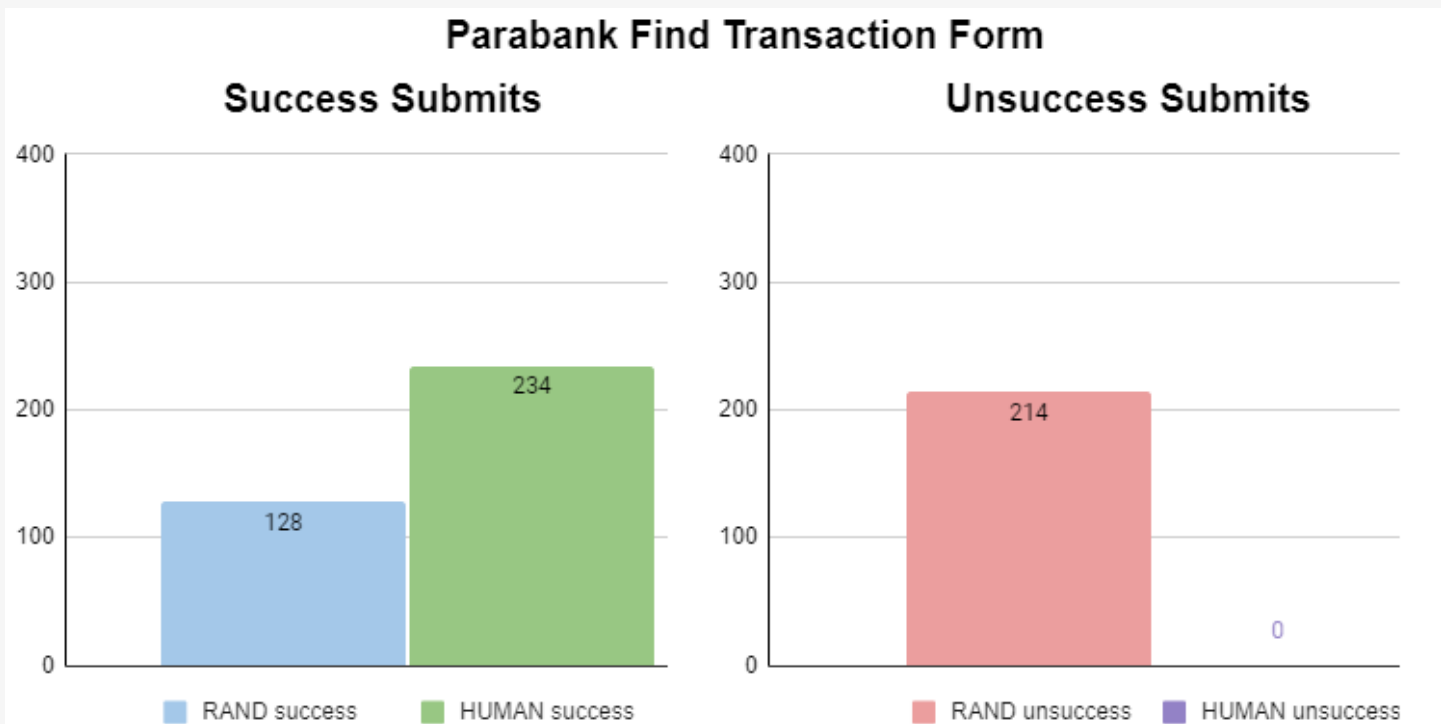
**FIND TRANSACTIONS**

---

Find by Amount:

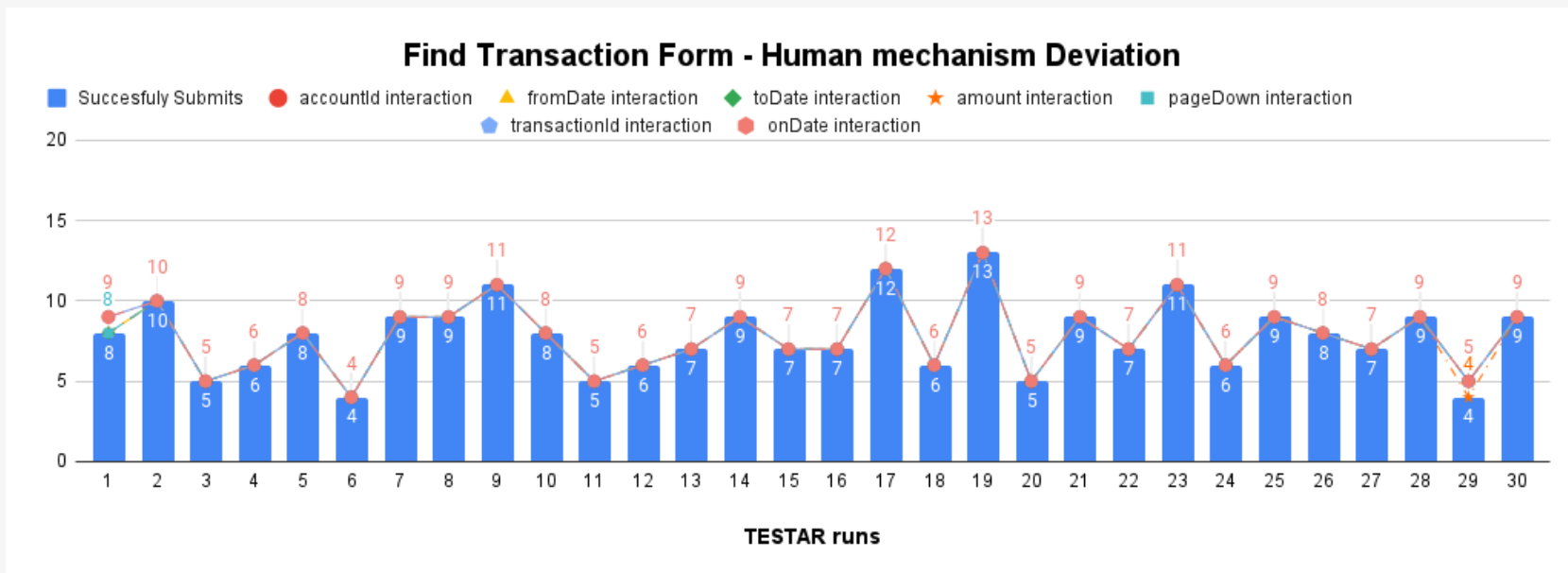
**FIND TRANSACTIONS**

# Parabank – Find Transactions Form Success and Unsuccess

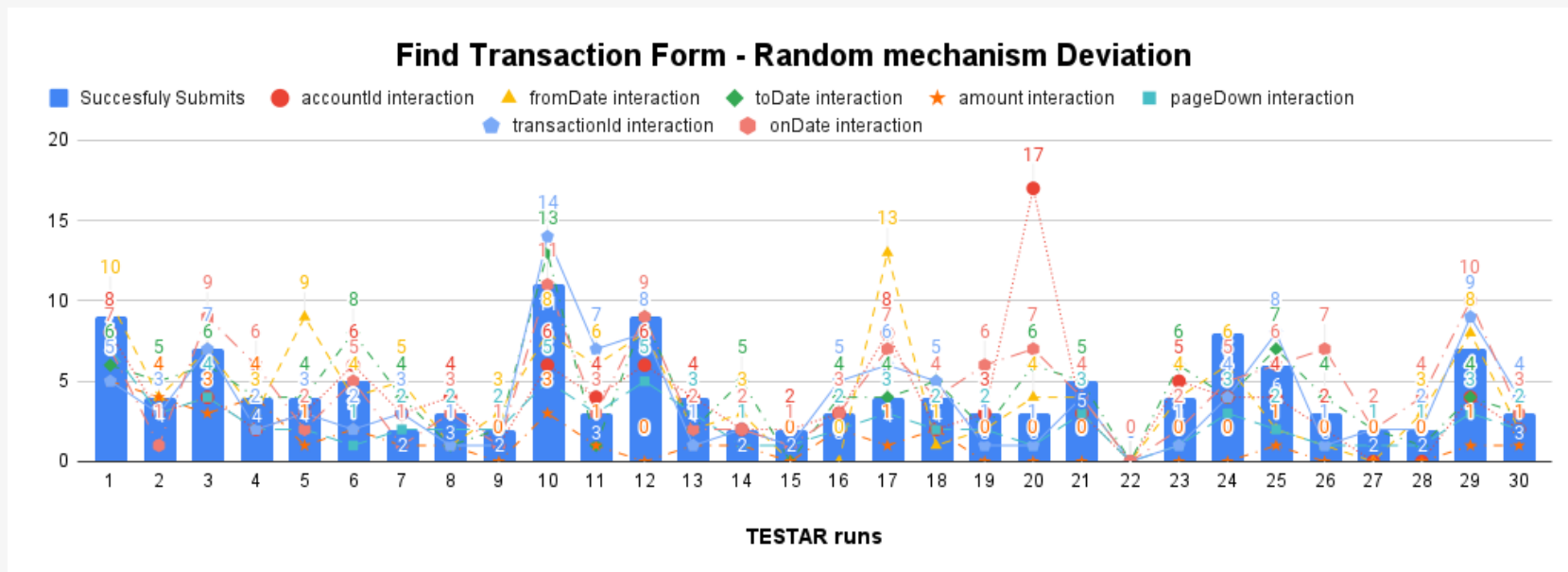




# Parabank – Find Transactions Form Interactions



# Parabank – Find Transactions Form Interactions





# Parabank – Request Loan Form

**Apply for a Loan**

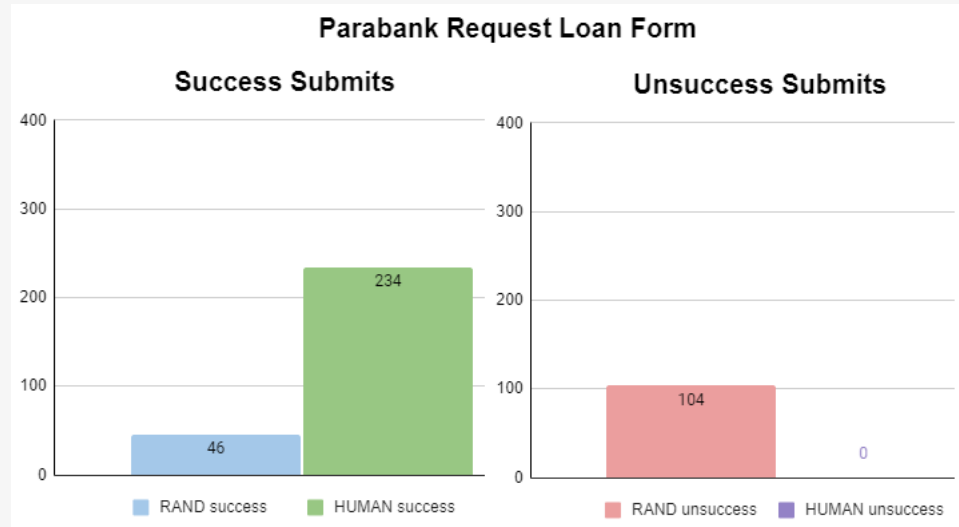
Loan Amount: \$

Down Payment: \$

From account #:

**APPLY NOW**

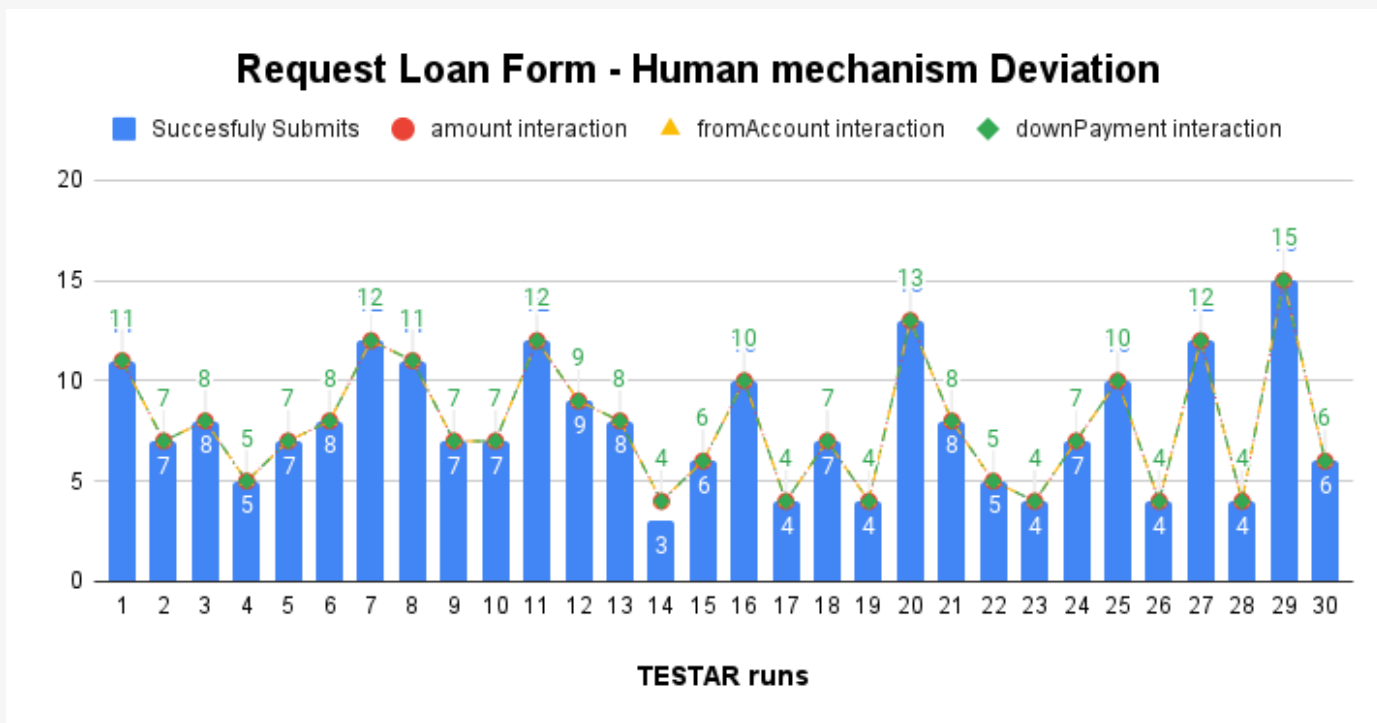
# Parabank – Request Loan Form Success and Unsuccess







# Parabank – Request Loan Form Interactions





# Parabank – Request Loan Form Interactions

