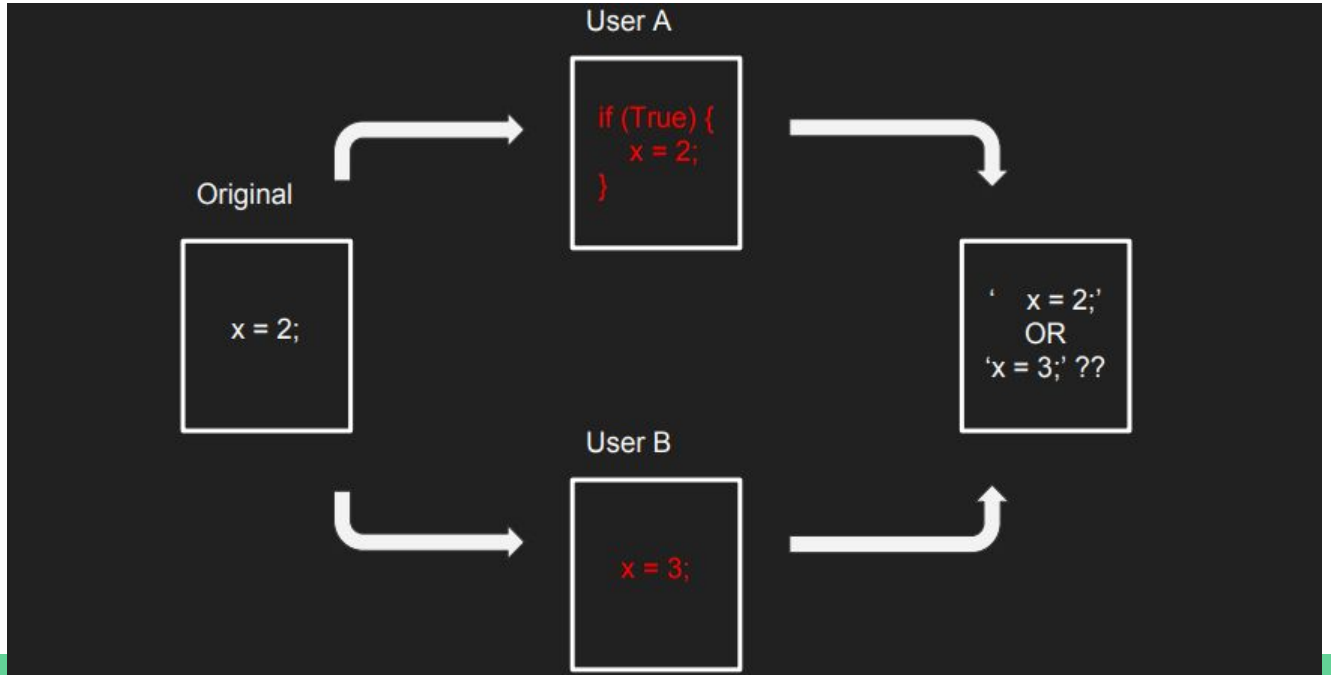
Smerge: Smarter Merge Conflict Resolutions

Alva Wei (alvawei)
Jediah Conachan (jediah6)
Kenji Nicholson (kenjilee)
Steven Miller (stevenm62)
Sungmin Rhee (srhee4)

Motivation

- Problem: Version control systems use line-based analysis to detect merge conflicts which results in many unnecessary manual merges
- Manual merges = valuable developer time = more money spent on projects



Risks & Challenges So Far

- Unsatisfactory merge would increase the user's workload
  - User would need to revert the change and manually resolve merge

- Balance between the amount of user input and the quality of merge
  - Minimize user input and maximize quality of merge

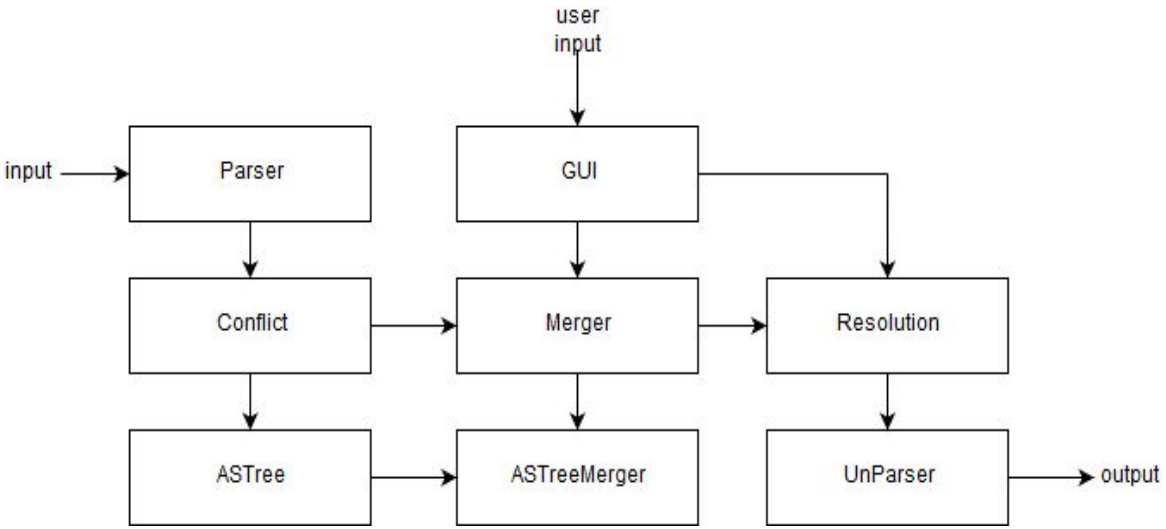
Goals

- Make handling merge conflicts easier and require less time/money
- Find a middle ground between automating merges and accepting input
- Make giving input intuitive and faster
    - Implement a GUI
- Reduce frequency of undesired behavior for merges that are automated

Current Approaches

- Two categories:
  - Conservative approaches that require lots of user input
    - Time consuming to fill out the user input
    - If they don’t require as much user input, usually few conflicts are resolved
    - Examples: BitBucket, Git ‘ignore whitespace’
  - Fully automated approaches that require no input
    - Creates defective merges with undesired interleavings
    - Some merges lead to undesired behavior
    - Examples: Conflerge

Approach



1. Input: git merge conflict via 2 conflicts
2. Parse input using JavaParser
3. Find conflicts in parsed code
4. Resolve with merger
    a. ASTs & Mergers
    b. GUI provides user input
5. Find resolution
6. Unparse back to code
7. Output: conflict resolution

Approach (cont.)

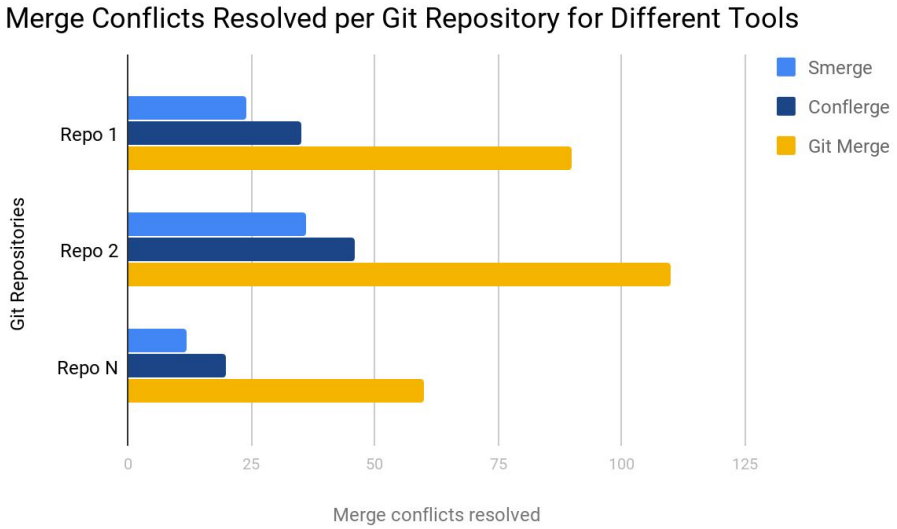
- GUI to take user inputs for better resolution
  - Auto-Solve Selection
  - Merge Confirmations w/ ‘Options’
  - Help tab opens user guide
  - Option tab opens auto-solve menu

Research

- Question: How successful is Smerge at reducing conflicts?
- Hypothesis: Smerge will reduce the number of merge conflicts experienced by the programmer
- Procedure
  - Gather many GitHub repositories and their respective historical data
  - From the historical data, look for merge commits that have two or more parents
  - Use git’s standard merge tools on the commits and record how many conflicts arise
  - Use other tools on the commits and record how many conflicts arise
  - Use Smerge’s merging algorithm and record how many conflicts arise
  - Compare and contrast the human resolution to the conflict with Smerge’s merge resolution

Research (cont.)

- Data Collection



- Conclusions
  - If Smerge has less conflicts than competitors, we will have succeeded
- Validity Threats
  - Sample selection bias
    - We select a small # of repos
  - GUI intuitiveness is subjective
    - Not everyone will like it

Related works

- Bitbucket
  - Resolves all conflicts by branch prioritizing
  - Possible undesirable merges
  - Works similarly to Github
  - Opposite to ours: Very conservative and requires manual action
- Conflerge
  - Token-based merge and AST-based merge for fine-grain analysis
  - Automatically resolves merge conflict about 50% of the time
  - Either succeed or fail silently
  - Some defective merges
  - The project we'll build off of