Clone and Similarity Management

Thomas Schmorleiz, May 12, 2015
The problem

- Software is often developed as a **set of variants** to
  - meet **conflicting requirements** and
  - implement **diverse specifications**

- E.g.:
  - Legal frameworks
  - Use cases
  - Management and marketing strategies
  - Cultural specifics
Real-world example

- Automotive Software
- More
  - user features
  - sensors
  - electronic aids
in higher-end models.
Running example

- **Human Resource Management System**

- **Features**
  - Model a company with departments and employees
  - Total salaries of all employees
  - Cut salaries of all employees
  - Depth of a company
Running example

- 3 target variants
  - 1. “basic”
    - Company, Total, Cut

```
1-- companies as lists of departments
2data Company = Company [Department]
3
4-- departments with names, departments and employees
5data Department = Department String [Department] [Employee]
6
7-- employees with names and salaries
8data Employee = Employee String Float
```
Running example

3 target variants

1. “basic”
   Company, **Total**, Cut

```haskell
30 totalSalaries :: Company -> Float
31 totalSalaries (Company depts) = sum$ map totalDept depts
32    where
33        totalDept (Department _ depts emps) =
34            (sum$ map totalDept depts) + (sum$ map totalEmp emps)
35        totalEmp (Employee _ s) = s
```
Running example

- 3 target variants
  - 1. “basic”
    - Company, Total, **Cut**

```haskell
cutSalaries :: Company -> Company

cutSalaries (Company depts) = Company (map cutDept depts)

    where

cutDept (Department n depts emps) =

    cutEmp (Employee n s) = Employee n (s / 2)
```
Running example

- 3 target variants
  - 2. Implementation with “advanced cutting”
    - Company, Total, Cut
    - Cloned from “basic”
Running example

- **3 target variants**
  - 2. Implementation with “advanced cutting”
    - Company, Total, Cut
    - Cloned from “basic”, and modified

```haskell
37 cutSalaries :: Company -> Float -> Company
38 cutSalaries (Company depts) factor = Company (map cutDept depts)
   where
39     cutDept (Department n depts emps) =
40                 Department n (map cutDept depts) (map cutEmp emps)
41     cutEmp (Employee n s) = Employee n (s / factor)
```
Running example

- 3 target variants
  - 3. Implementation with “with depth”
    - Company, Total, Cut, Depth
    - Cloned from “basic”
Running example

- **3 target variants**
  - **2. Implementation with “with depth”**
    - Company, Total, **Cut**
      - **Cloned** from “basic” (**not** “advanced cutting”), and modified

```haskell
37 cutSalaries :: Company -> Float -> Company
38 cutSalaries (Company depts) x = Company (map cutDept depts)
39     where
40         cutDept (Department n depts emps) =
41                 Department n (map cutDept depts) (map cutEmp emps)
42         cutEmp (Employee n s) = Employee n (s / x)
```
Running example

3 target variants

3. Implementation with “with cutting”

Company, Total, Cut, Depth

44\[\text{depth :: Company -> Int}\]
45\[\text{depth \ (Company \ depts) = 1 + (maximum \$ \ map \ depthDept \ depts)}\]
46\[\text{where}\]
47\[\text{depthDept \ (Department \_ \ depts \ emps) =}\]
48\[1 + (maximum \$ \ (map \ depthDept \ depts) ++ \ (map \ depthEmp \ emps))\]
49\[\text{depthEmp e = 1}\]
First issues

1. For the Cut “advanced cutting” and “with depth” we should establish an equality
Evolution

- Say, the specification of the Model feature is updated

“Employees have a name and a salary”

“Employees have a name, a salary, and an address”
“Employees have a name, a salary, and an address”
“basic” implementation co-evolves

```haskell
1-- companies as lists of departments
2data Company = Company [Department]

4-- departments with names, departments and employees
5data Department = Department String [Department] [Employee]

7-- employees with names, salaries, and addresses
8data Employee = Employee String Float String
```
Evolution

- “Employees have a name, a salary, and an address”
- “basic” implementation co-evolves

```haskell
30 totalSalaries :: Company -> Float
31 totalSalaries (Company depts) = sum$ map totalDept depts
32    where
33        totalDept (Department _ depts emps) =
34            (sum$ map totalDept depts) + (sum$ map totalEmp emps)
35        totalEmp (Employee _ s _) = s
```
Evolution

"Employees have a name, a salary, and an address"

"basic" implementation co-evolves

```haskell
37 cutSalaries :: Company -> Company
38 cutSalaries (Company depts) = Company (map cutDept depts)
39     where
40         cutDept (Department n depts emps) =
41         Department n (map cutDept depts) (map cutEmp emps)
42         cutEmp (Employee n s a) = Employee n (s / 2) a
```
First issues

2. Other implementations need to adjust too
   - For Model and Total we need to restore equalities to “basic”
   - For Cut we need to increase similarities to “basic”
First issues, observations

- Cloning may lead to inconsistencies and divergence
- Certain operations need to be performed to diminish such issues:
  - Establish equality
  - Restore equality
  - Increase similarity
Common approaches

Two common approaches to develop a set of variants

1. “clone-and-own”, copy assets to create a new variants from an existing one

   - Flexible; independent developers; non-disruptive development

   - No process; fragments diverge over time, have to synchronized manually
Two common approaches to develop a set of variants

2. **Product line engineering**, build a platform of shared assets

- minimal redundancy; build-in change propagation
- High-risk migration process; less independent developers; no proactively adoptable
Idea

- Some middle ground is needed
- Should combine advantages of clone-and-own and PLE
- Should diminish their disadvantages
- Should allow cloning while providing means of variant, clone, similarity management and monitoring
Idea

- **Analyze** the similarities by extracting metadata throughout the history of a repository

- **Annotate** the similarities by expressing how they should be maintained further

- **Maintain** the similarities by automatic change propagation or manual actions
Previous work

- Virtual platform of metadata with cloning-related operator

In this context:
- extracted metadata = virtual platform
- propagation = implementation of propagate operator
Agenda

1. Similarity analysis
2. Similarity annotations
3. Similarity management tool support
   1. Similarity exploration
   2. Annotation inference
   3. Automatic change propagation
   4. Recommending and updating annotations
   5. Manual maintenance
4. Git integration
5. Case study: 101haskell
Similarity Analysis
Similarity analysis

- Automated process to extract metadata from the history of a repository
- History as sequence of commits
- Use APIs to access local repository
- Store extracted metadata in a database for later use
Extraction:
1. Variants
Variant extraction

- Organizing variants
  - Dedicated repos
  - Dedicated branches in one repo
  - Dedicated folders in one branch
Variant extraction

- **Input**: Set of possible root folders $ps$
- For every path $p$ in $ps$, and every commit $c$:
  - check whether $p$ exists at $c$
- For the first hit (regarding the order given by $p$):
  - extract names of subfolders as variant names
Variant extraction

- Can detect renaming: If
  - the majority of files in folder \( v_1 \) at \( c_1 \) where moved to \( v_2 \) at \( c_2 \)
  - \( v_2 \) did not exist before, and
  - \( v_1 \) no longer exists

- Can detect splits:
  - Same as renaming but \( v_1 \) continues to exist
Implementation

- **Language:** Python, **Git API:** GitPython

```python
def extract_variants(repo, branch_name):
    repo = Repo(repo_path)
    commits = repo.iter_commits(branch_name)
    extract_variant_variants(repo_path, variation_dirs, repo, db, commits)
```

- **Iterate commits**

- **Extract variants**
```python
def extract(repo_path, variation_dirs, repo, db, commits):
    last_commit = None
    # extract variants from each commit
    for commit in commits:
        (current_renamings, current_variants) = from_commit(commit, last_commit)
        all_variants[commit.hexsha] = current_variants
        all_renamings[commit.hexsha] = current_renamings
        last_commit = commit

    document = {}
    document["variants"] = all_variants
    document["renamings"] = all_renamings
    # print document
    db.repos.update({"path": repo_path}, {"$set" : document})
```
Implementation

```python
def extract(repo_path, variation_dirs, repo, db, commits):
    last_commit = None
    # extract variants from each commit
    for commit in commits:
        (current_renamings, current_variants) = from_commit(commit, last_commit)
        all_variants[commit.hexsha] = current_variants
        all_renamings[commit.hexsha] = current_renamings
        last_commit = commit

    document = {}
    document["variants"] = all_variants
    document["renamings"] = all_renamings
    #print document
    db.repos.update({"path": repo_path}, {"$set" : document})
```
Implementation

```python
# find the current variation directory

def find_variation_dir():
    for candidate in variation_dirs:
        full_candidate = os.path.join(repo_path, candidate)
        if os.path.isdir(full_candidate):
            return candidate
    return None

# extract all variants at commit

def from_commit(commit, last_commit):
    print("> " + commit.hexsha)
    # first, checkout the commit
    repo.git.checkout(commit.hexsha)
    # find the current variation directory
    variation_dir = find_variation_dir()
    full_variation_dir = os.path.join(repo_path, variation_dir)
    variants = filter(lambda x: x[0] != '.', os.listdir(full_variation_dir))

Code for detecting renaming omitted
```
# find the current variation directory

def find_variation_dir():
    for candidate in variation_dirs:
        full_candidate = os.path.join(repo_path, candidate)
        if os.path.isdir(full_candidate):
            return candidate
    return None

# extract all variants at commit

def from_commit(commit, last_commit):
    print("> " + commit.hexsha)
    # first, checkout the commit
    repo.git.checkout(commit.hexsha)
    # find the current variation directory
    variation_dir = find_variation_dir()
    full_variation_dir = os.path.join(repo_path, variation_dir)
    variants = filter(lambda x: x[0] != '.', os.listdir(full_variation_dir))
Implementation

```python
# find the current variation directory

def find_variation_dir():
    for candidate in variation_dirs:
        full_candidate = os.path.join(repo_path, candidate)
        if os.path.isdir(full_candidate):
            return candidate
    return None

# extract all variants at commit

def from_commit(commit, last_commit):
    print('>' + commit.hexsha
    # first, checkout the commit
    repo.git.checkout(commit.hexsha)
    # find the current variation directory
    variation_dir = find_variation_dir()
    full_variation_dir = os.path.join(repo_path, variation_dir)
    variants = filter(lambda x: x[0] != '.', os.listdir(full_variation_dir))

Code for detecting renaming omitted
Implementation

```python
15    # find the current variation directory
16    def find_variation_dir():
17        for candidate in variation_dirs:
18            full_candidate = os.path.join(repo_path, candidate)
19            if os.path.isdir(full_candidate):
20                return candidate
21        return None
22
23    # extract all variants at commit
24    def from_commit(commit, last_commit):
25        print ("> " + commit.hexsha
26        # first, checkout the commit
27        repo.git.checkout(commit.hexsha)
28        # find the current variation directory
29        variation_dir = find_variation_dir()
30        full_variation_dir = os.path.join(repo_path, variation_dir)
31        variants = filter(lambda x: x[0] != '.', os.listdir(full_variation_dir))
```

Code for detecting renaming omitted
Example

Variants

```json
{
    "7528896517eb0c42fbd6f486e7a4866cae0c0d8" : [ "basic" ],
    "02fca68e0cb533e0663e3c8d0f3567e171b2111e" : [ "advanced-cutting", "basic" ],
    "f6dad913b0901f3eae767d556e44f32b8a9a723f" : [ "advanced-cutting", "basic" ],
    "da6ad130fbf6cac3e2d21d96349fe60535b42ac5" : [ "advanced-cutting", "basic", "with-depth" ]
}
```

Renaming

{}
Extraction:
2. Fragment snapshots
Fragment snapshot extraction

- Fragment consist of **consecutive lines of code**
- It corresponds to a **hand-selected node** in the AST of the file
- Within in file, fragments are identified by a **classifier/name pair**
  - Classifiers:
    - Haskell: function, data, type
    - Java: class, method, field
- Fact extractors developed for wide range of languages
- Fragment snapshots are bound to a commit point
Implementation

Configuration file for extraction technology per language

```
{
    "fragments-tech": {
        "hs": {
            "language": "Haskell",
            "runner": "runhaskell",
            "extractor": "technologies/HsFactExtractor/HsFactExtractor.hs",
            "locator": "technologies/HsFragmentLocator/HsFragmentLocator.hs",
            "line-pp": "technologies/HsTokenizer/TokenLinePrinter.hs",
            "classifier_blacklist": []
        },
        ...
    }
}
```
def extract(repo_path, tech_path, min_size, variation_dirs, repo, db, commits):

    print 'Extracting from ' + str(len(commits)) + ' commits.'

    # lookup variations directories and variations
    repo_data = db.repos.find({'path': repo_path})[0]
    all_variants = repo_data['variants']

    # extract fragment snapshots from each commit
    for idx, commit in enumerate(commits):
        from_commit(commit, variation_dirs, all_variants[commit.hexsha], db)
def from_commit(commit, variation_dirs, variants, db):
    global cache
    result = []
    print('> ' + commit.hexsha)
    # first, checkout the commit
    repo.git.checkout(commit.hexsha)
    # find the current variation directory
    variation_dir = find_variation_dir(repo_path, variation_dirs)
    full_variation_dir = os.path.join(repo_path, variation_dir)
    # extract for all current variations
    for variant in variants:
        full_current_variation_dir = os.path.join(full_variation_dir, variant)
        # extract for all known extensions
        for extension in config:
            # find all files with the current extension
            found_files = list(find_files(
                path=full_current_variation_dir,
                match=Match(filetype='f',
                             name='*. ' + extension)
            ))
```python
def from_commit(commit, variation_dirs, variants, db):
    global cache
    result = []
    print('>' + commit.hexsha
    # first, checkout the commit
    repo.git.checkout(commit.hexsha)
    # find the current variation directory
    variation_dir = find_variation_dir(repo_path, variation_dirs)
    full_variation_dir = os.path.join(repo_path, variation_dir)
    # extract for all current variations
    for variant in variants:
        full_current_variation_dir = os.path.join(full_variation_dir, variant)
        # extract for all known extensions
        for extension in config:
            # find all files with the current extension
            found_files = list(find_files(
                path=full_current_variation_dir,
                match=Match(filetype='f',
                             name='*' + extension)
            ))
```
Implementation

```python
for found_file in found_files:
    rel_path = os.path.relpath(found_file, full_current_variation_dir)
    # else extract
    fragment_snapshots = get_fragment_snapshots(found_file, rel_path,
        config[extension]['runner'],
        config[extension]['extractor'],
        config[extension]['classifier_blacklist'])
    for idx, fragment_snapshot in enumerate(fragment_snapshots):
        # extract line range and enrich fragment snapshot
        line_range = get_line_range(found_file, rel_path, fragment_snapshot,
            config[extension]['runner'],
            config[extension]['locator'])
        fragment_snapshots[idx]['language'] = config[extension]['language']
        fragment_snapshots[idx]['from'] = line_range['from']
        fragment_snapshots[idx]['to'] = line_range['to']
        fragment_snapshots[idx]['repo_path'] = repo_path
```
for found_file in found_files:
    rel_path = os.path.relpath(found_file, full_current_variation_dir)
    # else extract
    fragment_snapshots = get_fragment_snapshots(found_file, rel_path,
        config[extension]['runner'],
        config[extension]['extractor'],
        config[extension]['classifier_blacklist'])
    for idx, fragment_snapshot in enumerate(fragment_snapshots):
        # extract line range and enrich fragment snapshot
        line_range = get_line_range(found_file, rel_path, fragment_snapshot,
            config[extension]['runner'],
            config[extension]['locator'])
        fragment_snapshots[idx]['language'] = config[extension]['language']
        fragment_snapshots[idx]['from'] = line_range['from']
        fragment_snapshots[idx]['to'] = line_range['to']
        fragment_snapshots[idx]['repo_path'] = repo_path
for found_file in found_files:
    rel_path = os.path.relpath(found_file, full_current_variation_dir)
    # else extract
    fragment_snapshots = get_fragment_snapshots(found_file, rel_path,
        config[extension]['runner'],
        config[extension]['extractor'],
        config[extension]['classifier_blacklist'])
    for idx, fragment_snapshot in enumerate(fragment_snapshots):
        # extract line range and enrich fragment snapshot
        line_range = get_line_range(found_file, rel_path, fragment_snapshot,
            config[extension]['runner'],
            config[extension]['locator'])
        fragment_snapshots[idx]['language'] = config[extension]['language']
        fragment_snapshots[idx]['from'] = line_range['from']
        fragment_snapshots[idx]['to'] = line_range['to']
        fragment_snapshots[idx]['repo_path'] = repo_path
for found_file in found_files:
    rel_path = os.path.relpath(found_file, full_current_variation_dir)
    # else extract
    fragment_snapshots = get_fragment_snapshots(found_file, rel_path,
    config[extension]['runner'],
    config[extension]['extractor'],
    config[extension]['classifier_blacklist'])
    for idx, fragment_snapshot in enumerate(fragment_snapshots):
        # extract line range and enrich fragment snapshot
        line_range = get_line_range(found_file, rel_path, fragment_snapshot,
        config[extension]['runner'],
        config[extension]['locator'])
        fragment_snapshots[idx]['language'] = config[extension]['language']
        fragment_snapshots[idx]['from'] = line_range['from']
        fragment_snapshots[idx]['to'] = line_range['to']
        fragment_snapshots[idx]['repo_path'] = repo_path
def get_fragment_snapshots(abs_path, rel_path, runner, extractor_path, classifier_blacklist):
    extractor_path = os.path.join(tp, extractor_path)
    myinput = open(abs_path)
    process_args = [runner, extractor_path]
    result = Popen(process_args, stdin=myinput, stdout=PIPE).communicate()[0]
    result = json.loads(result)

def collect_fragments(level):
    if 'fragments' not in level:
        return []
    else:
        fs = []
        for f in level['fragments']:
            if f['classifier'] not in classifier_blacklist:
                f_doc = {
                    'classifier': f['classifier'],
                    'name': f['name']
                }
                fs.append(f_doc)
                fs.extend(collect_fragments(f))
        return fs
    return {'fragments': collect_fragments(result)}
def get_line_range(abs_path, rel_path, fragment_snapshot, runner, locator_path):
    locator_path = os.path.join(tp, locator_path)
    fragment_snapshot_path = os.path.join(
        fragment_snapshot['classifier'],
        fragment_snapshot['name'])
    myinput = open(abs_path)
    result = Popen([runner, locator_path, fragment_snapshot_path],
                    stdin=myinput,
                    stdout=PIPE).communicate()[0]
    return json.loads(result)
Example

- Fragment snapshot

```haskell
-- companies as lists of departments
data Company = Company [Department]

{ "_id" : ObjectId("554a5c6c353ec807909ba32c"),
  "variant" : "basic",
  "relative_path" : "Main.hs",
  "name" : "Company",
  "classifier" : "data",
  "from" : 1,
  "to" : 2,
  "language" : "Haskell",
  "sha" : "f6dad913b0901f3eae767d556e44f32b8a9a723f",
  "repo_path" : "/Users/tschmorleiz/Desktop/msr/implementations/"
}
```
Fragment snapshot extraction
Extraction:
3. Fragments
Fragment extraction

- Fragment snapshots are bound to commits
- A fragment consists of a series of its snapshots
- To extract a fragment we need to link its snapshots
- Need to be able to handle
  - A. Renaming at variant, file, fragment level
  - B. Content changes
  - C. Interactions of renaming and content changes
Implementation, abstract

- For a fragment snapshot $fs_i$ at commit point $c_i$
- Check each snapshot $fs_{i-1}$ at $c_{i-1}$ in the same file as $fs_i$:
  1. **No change**: Link snapshots if snapshot can be found with exact same content
  2. **Pure renaming**: Link snapshots if only the name was changed
  3. **Mass renaming**: Link snapshots if contents match except for any name of fragments in the same file
  4. **Pure content editing**: Link snapshots if classifier and name matches
Fragment extraction

- Order of cases is used when trying to find a linkable fragment and expresses priority

- With each case comes a set of flags for the linked snapshots
  - is_new
  - is_changed
  - is_renamed

- If no snapshot to link to can be found, extend search space to variant level

- If still no linkable snapshot can be found, snapshot marks the creation of a fragment
Commit traversal

- We do not consider branches as a means of organizing variants

- However we consider them when traversing

- Scenario: fragment f was added at c1

- At c3 and fragment snapshot $f_{s_c3}$:
  - Just looking at c2 (last commit via timestamp) fails
  - Look at both c1 and c2 for fragment snapshots to link to $f_{s_c3}$!
Example

Fragment

```json
{
  "_id" : ObjectId("554a5bdf353ec805268f2ef1"),
  "fragment_snapshot_ids" : [
    ObjectId("554a5bd9353ec805268f2ec4"),
    ObjectId("554a5bdc353ec805268f2ed0"),
    ObjectId("554a5bdc353ec805268f2edc")
  ],
  "repo_path" : "/Users/tschmorleiz/Desktop/msr/implementations/
}
```
Fragment extraction

fragment snapshot

- fragment
  - is_new
  - is_changed
  - is_renamed
Extraction:
4. Similarity Snapshots
A similarity snapshot is a **relation** of **two fragment snapshot** at a **commit point**

As a **similarity measure** we use “diff ratio” after pretty-printing tokens into many lines (proposed by Cordy et al.)

We therefore disregard similarities solely based on **whitespace**

To reduce number of relations stored, user provides a **similarity threshold**
We implemented a token pretty printer various languages.

```haskell
-- companies as lists of departments
data Company = Company [Department]

main = getContents >>= (putStrLn ppTokens.lexerPass0)

ppTokens :: [PosToken] -> String
ppTokens pts = join "\n" $ map (snd.snd) $ filter ((/=Whitespace).fst) pts
```

Input

```
1-- companies as lists of departments
2data Company = Company [Department]
```

Output

```
1--
2 companies as lists of departments
3
4data
5Company
6=
7 Company
8[
9Department
10]
```
Pretty Printing

Configuration file for pretty printing technology per language

```json
{
    "fragments-tech": {
        "hs": {
            "language": "Haskell",
            "runner": "runhaskell",
            "extractor": "technologies/HsFactExtractor/HsFactExtractor.hs",
            "locator": "technologies/HsFragmentLocator/HsFragmentLocator.hs",
            "line-pp": "technologies/HsTokenizer/TokenLinePrinter.hs",
            "classifier_blacklist": []
        }
    }
}
```
def from_commit(commit, all_fragment_snapshots, db, fragmentTracks, track_lookup):
    print '> ' + commit.hexsha
    variation_dir = find_variation_dir(repo_path, variation_dirs)
    # lookup current snapshots
    fragment_snapshots = filter(lambda f: f['sha'] == commit.hexsha, all_fragment_snapshots)
    print ' > ' + str(len(fragment_snapshots)) + ' fragments.'
    # if either snapshot marks a change, compute diff ratio, otherwise copy
    for fragment_snapshot in fragment_snapshots:
        if fragment_snapshot['is_new'] or fragment_snapshot['is_changed']:
            for other_fragment_snapshot in fragment_snapshots:
                from_pair(fragment_snapshot, other_fragment_snapshot, variation_dirs)
        else:
            for other_fragment_snapshot in fragment_snapshots:
                if other_fragment_snapshot['is_new'] or other_fragment_snapshot['is_changed']:
                    from_pair(fragment_snapshot, other_fragment_snapshot, variation_dirs)
                else:
                    copy_from_pair(...)

Implementation
def from_commit(commit, all_fragment_snapshots, db, fragmentTracks, track_lookup):
    print '>', commit.hexsha

    variation_dir = find_variation_dir(repo_path, variation_dirs)

    # lookup current snapshots
    fragment_snapshots = filter(lambda f: f['sha'] == commit.hexsha, all_fragment_snapshots)
    print '>', str(len(fragment_snapshots)) + ' fragments.'

    # if either snapshot marks a change, compute diff ratio, otherwise copy
    for fragment_snapshot in fragment_snapshots:
        if fragment_snapshot['is_new'] or fragment_snapshot['is_changed']:
            for other_fragment_snapshot in fragment_snapshots:
                from_pair(fragment_snapshot, other_fragment_snapshot, variation_dirs)
        else:
            for other_fragment_snapshot in fragment_snapshots:
                if other_fragment_snapshot['is_new'] or other_fragment_snapshot['is_changed']:
                    from_pair(fragment_snapshot, other_fragment_snapshot, variation_dirs)
                else:
                    copy_from_pair(...
def from_commit(commit, all_fragment_snapshots, db, fragmentTracks, track_lookup):
    print '> ' + commit.hexsha
    variation_dir = find_variation_dir(repo_path, variation_dirs)
    # lookup current snapshots
    fragment_snapshots = filter(lambda f: f['sha'] == commit.hexsha, all_fragment_snapshots)
    print ' > ' + str(len(fragment_snapshots)) + ' fragments.'
    # if either snapshot marks a change, compute diff ratio, otherwise copy
    for fragment_snapshot in fragment_snapshots:
        if fragment_snapshot['is_new'] or fragment_snapshot['is_changed']:
            for other_fragment_snapshot in fragment_snapshots:
                from_pair(fragment_snapshot, other_fragment_snapshot, variation_dirs)
        else:
            for other_fragment_snapshot in fragment_snapshots:
                if other_fragment_snapshot['is_new'] or other_fragment_snapshot['is_changed']:
                    from_pair(fragment_snapshot, other_fragment_snapshot, variation_dirs)
                else:
                    copy_from_pair(...)
Implementation

```python
def from_pair(fragment_snapshot, other_fragment_snapshot, variation_dirs):
    if fragment_snapshot['_id'] == other_fragment_snapshot['_id']:
        return None

    ratio = diff_ratio(fragment_snapshot, other_fragment_snapshot, variation_dirs)

    document = {}
    document['sha'] = commit.hexsha
    document['repo_path'] = rp
    document['source_id'] = other_fragment_snapshot['_id']
    document['source_variant'] = other_fragment_snapshot['variant']
    document['target_id'] = fragment_snapshot['_id']
    document['target_variant'] = fragment_snapshot['variant']

    db.similaritySnapshots.insert(document)
```
def from_pair(fragment_snapshot, other_fragment_snapshot, variation_dirs):
    if fragment_snapshot['_id'] == other_fragment_snapshot['_id']:
        return None
    ratio = diff_ratio(fragment_snapshot, other_fragment_snapshot, variation_dirs)

    document = {
        'sha': commit.hexsha,
        'repo_path': rp,
        'source_id': other_fragment_snapshot['_id'],
        'source_variant': other_fragment_snapshot['variant'],
        'target_id': fragment_snapshot['_id'],
        'target_variant': fragment_snapshot['variant']
    }
    db.similaritySnapshots.insert(document)
def diff_ratio(f1, f2, variation_dirs):
    content_f1 = get_fragment_snapshot_content(f1, variation_dirs)
    content_f2 = get_fragment_snapshot_content(f2, variation_dirs)
    return difflib.SequenceMatcher(None, content_f1, content_f2).ratio()
def get_fragment_snapshot_content(fragment, variation_dirs):
    checkout(fragment['sha'])
    variation_dir = find_variation_dir(rp, variation_dirs)
    path = os.path.join(repo_path, variation_dir, fragment['variant'], fragment['relative_path'])
    extension = os.path.splitext(fragment['relative_path'])[1][1:]
    with open(path) as f:
        content = ''.join(f.readlines()[fragment['from'] - 1:fragment['to']])
        if extension in config and 'line-pp' in config[extension]:
            temp_f = tempfile.NamedTemporaryFile(delete=False)
            temp_f.write(content)
            temp_f.close()
            line_pp_path = os.path.join(tp, config[extension]['line-pp'])
            result = Popen([config[extension]['runner'], line_pp_path],
                            stdin=open(temp_f.name), stdout=PIPE).communicate()[0]
            content = result
            os.unlink(temp_f.name)
    return content
Example

- Similarity snapshot

```json
{
    "id" : ObjectId("554a5caf353ec807909ba36a"),
    "target_id" : ObjectId("554a5c6c353ec807909ba331"),
    "source_id" : ObjectId("554a5c6c353ec807909ba32b"),
    "diff_ratio" : 0.9425742574257425,
    "sha" : "f6dad913b0901f3eae767d556e44f32b8a9a723f",
    "repo_path" : "/Users/tschmorleiz/Desktop/msr/implementations/
}
```
Similarity snapshot extraction
Extraction:
4. Similarity Evolutions
Similarity evolution extraction

- Similarity evolutions are series of similarity snapshots.
- To extract an evolution we extract similarity snapshots between respective fragment snapshots that all belong to the same fragment.
Example

Similarity evolution

```json
{
  "_id": ObjectId("554a5cb0353ec807909ba395"),
  "similarity_ids": [
    ObjectId("554a5caf353ec807909ba35f"),
    ObjectId("554a5caf353ec807909ba36b"),
    ObjectId("554a5cb0353ec807909ba37d")
  ],
  "first_diff_ratio": 1,
  "last_diff_ratio": 0.9425742574257425,
  "min_diff_ratio": 0.9425742574257425,
  "max_diff_ratio": 1
}
```
Similarity evolution extraction
To later compute the applicability of annotations for similarity evolutions, we assign each evolution a category:

- **Always Equal** holds similarity evolutions that have always been equalities, that is, the similarity value was always 1.

- **Converge to Equal** holds similarity evolutions where the similarity value was once below 1 but then increased to 1 at the HEAD of the branch.

- **Diverge from Equal** holds similarity evolutions where the similarity value was once 1 but then decreased to below 1 at the HEAD of the branch.

- **Always Non-equal** holds similarity evolutions where the similarity value was constant or changing but always below 1.
def get_category(last_diff_ratio, min_diff_ratio, max_diff_ratio):
    if min_diff_ratio == 1 and max_diff_ratio == 1:
        return 'always_equal'
    if min_diff_ratio < 1 and last_diff_ratio == 1:
        return 'converge_to_equal'
    if max_diff_ratio == 1 and last_diff_ratio < 1:
        return 'diverge_from_equal'
    if max_diff_ratio < 1:
        return 'always_nonequal'
Extraction: Updating metadata
Similarity evolution categorization

- For every new commit we have to update the metadata:
  - Extract new variant names
  - Extract new fragment snapshots
  - Create new or extend existing fragments
  - Extract new similarity snapshots
  - Create new or extend existing similarity evolutions
Similarity Annotations
We identify the following seven categories, each stating a maintenance invariant or task for the similarity

- **Invariants**

  - *Maintain Equality* by automatic three-way merge when one or both fragments of the similarity evolution change and potentially by manual conflict resolution.
Maintain Equality

In “basic”:

```
4  -- departments with names, departments and employees
5  data Department = Department String [Department] [Employee]
```

In “advanced-cutting”:

```
4  -- departments with names, departments and employees
5  data Department = Department String [Department] [Employee]
```
Annotation categories

- Invariants
  - Maintain Similarity by manual actions when the similarity changes.
Maintain Similarity

In “basic”:

37 \texttt{cutSalaries :: Company -> Company} \\
38 \texttt{cutSalaries (Company depts) = Company (map cutDept depts)} \\
39 \hspace{1em} \texttt{where} \\
40 \quad \texttt{cutDept (Department n depts emps) =} \\
41 \quad \quad \texttt{Department n (map cutDept depts) (map cutEmp emps)} \\
42 \quad \texttt{cutEmp (Employee n s) = Employee n (s / 2)} \\

In “advanced-cutting”:

37 \texttt{cutSalaries :: Company -> Float -> Company} \\
38 \texttt{cutSalaries (Company depts) factor = Company (map cutDept depts)} \\
39 \hspace{1em} \texttt{where} \\
40 \quad \texttt{cutDept (Department n depts emps) =} \\
41 \quad \quad \texttt{Department n (map cutDept depts) (map cutEmp emps)} \\
42 \quad \texttt{cutEmp (Employee n s) = Employee n (s / factor)}
Annotation categories

- Invariants
  - Ignore Similarity by not reporting the similarity anymore to the user.
In “basic”:

```haskell
30 totalSalaries :: Company -> Float
31 totalSalaries (Company depts) = sum $ map totalDept depts
32     where
33         totalDept (Department _ depts emps) =
34             (sum $ map totalDept depts) + (sum $ map totalEmp emps)
35         totalEmp (Employee _ s _) = s
```

In “basic”:

```haskell
37 cutSalaries :: Company -> Company
38 cutSalaries (Company depts) = Company (map cutDept depts)
39     where
40         cutDept (Department n depts emps) =
41             Department n (map cutDept depts) (map cutEmp emps)
42         cutEmp (Employee n s a) = Employee n (s / 2) a
```
Tasks

- *Restore Equality* by automatically propagating changes from one fragment to the other if a direction of propagation is defined, otherwise by automatic three-way merge and potentially by manual conflict resolution.
In "basic" after adjusting to new spec:

```haskell
7  -- employees with names, salaries, and addresses
8  data Employee = Employee String Float String
```

Restore Equality

In "advanced-cutting”:

```haskell
7  -- employees with names and salaries
8  data Employee = Employee String Float
```
Annotation categories

Tasks

- *Establish Equality* by manual actions on fragments that have never been equal.
Establish Equality

In “advanced-cutting”:

```haskell
37 cutSalaries :: Company -> Float -> Company
38 cutSalaries (Company depts) factor = Company (map cutDept depts)
39     where
40         cutDept (Department n depts emps) =
41             Department n (map cutDept depts) (map cutEmp emps)
42         cutEmp (Employee n s) = Employee n (s / factor)
```

In “with-depth”:

```haskell
37 cutSalaries :: Company -> Float -> Company
38 cutSalaries (Company depts) x = Company (map cutDept depts)
39     where
40         cutDept (Department n depts emps) =
41             Department n (map cutDept depts) (map cutEmp emps)
42         cutEmp (Employee n s) = Employee n (s / x)
```
Annotation categories

- Tasks
  - *Increase Similarity* by manual actions.
Maintain Similarity

In “basic” after adjusting to new spec:

\[
\text{cutSalaries :: Company -> Company}
\]
\[
\text{cutSalaries (Company depts) = Company (map cutDept depts)}
\]
\[
\text{where}
\]
\[
\text{cutDept (Department n depts emps) =}
\]
\[
\text{Department n (map cutDept depts) (map cutEmp emps)}
\]
\[
\text{cutEmp (Employee n s a) = Employee n (s / 2) a}
\]

In “advanced-cutting”:

\[
\text{cutSalaries :: Company -> Float -> Company}
\]
\[
\text{cutSalaries (Company depts) factor = Company (map cutDept depts)}
\]
\[
\text{where}
\]
\[
\text{cutDept (Department n depts emps) =}
\]
\[
\text{Department n (map cutDept depts) (map cutEmp emps)}
\]
\[
\text{cutEmp (Employee n s a) = Employee n (s / factor)}
\]
Annotation categories

- Tasks
  - *Restore Similarity* by manual actions until a target similarity value is reached.
Maintain Similarity

In “basic”:

```
37 cutSalaries :: Company -> Company
38 cutSalaries (Company depts) = Company (map cutDept depts)
39     where
40         cutDept (Department n depts emps) =
41             Department n (map cutDept depts) (map cutEmp emps)
42         cutEmp (Employee n s) = Employee n (s / 2)
```

In “advanced-cutting”:

```
37 cutSalaries :: Company -> Float -> Company
38 cutSalaries (Company depts) factor = Company (map cutDept depts)
39     where
40         cutDept (Department n depts emps) =
41             Department n (map cutDept depts) (map cutEmp emps)
42         cutEmp (Employee n s) = Employee n (s / factor)
```
Maintain Similarity

In “basic” after adjusting to new spec:

```haskell
37 cutSalaries :: Company -> Company
38 cutSalaries (Company depts) = Company (map cutDept depts)
39    where
40        cutDept (Department n depts emps) =
41            Department n (map cutDept depts) (map cutEmp emps)
42        cutEmp (Employee n s a) = Employee n (s / 2) a
```

In “advanced-cutting”:

```haskell
37 cutSalaries :: Company -> Float -> Company
38 cutSalaries (Company depts) factor = Company (map cutDept depts)
39    where
40        cutDept (Department n depts emps) =
41            Department n (map cutDept depts) (map cutEmp emps)
42        cutEmp (Employee n s) = Employee n (s / factor)
```
Annotation categories

○ Tasks

○ *Remove Equality* by manually making fragments unequal.
# Annotation applicabilities

<table>
<thead>
<tr>
<th></th>
<th>MAINTAIN EQUALITY</th>
<th>RESTORE EQUALITY</th>
<th>ESTABLISH EQUALITY</th>
<th>MAINTAIN SIMILARITY</th>
<th>RESTORE SIMILARITY</th>
<th>INCREASE SIMILARITY</th>
<th>IGNORE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALWAYS EQUAL</strong></td>
<td>AUTO</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>AUTO</td>
</tr>
<tr>
<td><strong>CONVERGE TO EQUAL</strong></td>
<td>AUTO</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>AUTO</td>
</tr>
<tr>
<td><strong>DIVERGE FROM EQUAL</strong></td>
<td>X</td>
<td>AUTO</td>
<td>X</td>
<td>MANUAL</td>
<td>MANUAL</td>
<td>MANUAL</td>
<td>AUTO</td>
</tr>
<tr>
<td><strong>ALWAYS NON-EQUAL</strong></td>
<td>X</td>
<td>X</td>
<td>MANUAL</td>
<td>MANUAL</td>
<td>MANUAL</td>
<td>MANUAL</td>
<td>AUTO</td>
</tr>
</tbody>
</table>
Annotation structure

- **similarity**: reference to the annotated similarity

- **category**: Name of the category assigned by the user or by updating the category automatically

- **intent**: User comment about the annotation, can be about who should perform manual tasks, what to change manually, etc.

- **direction**: Only for category Restore Equality, specifies which fragments changes should be overwritten by automatic change propagation
Similarity Management Tool Support
Similarity Exploration
Annotators

- To provide two different annotators
- Usage depends on how the user want to explore the similarities
Commit-centric annotator

- Tailored for the use case where commits should be inspected one by one
Commit-centric annotator
Commit-centric annotator
Commit-centric annotator
Commit-centric annotator
Commit-centric annotator
Variant-centric annotator

- Tailored for the use case where variants should be excepted at HEAD
Variant-centric annotator
Variant-centric annotator
Variant-centric annotator
Variant-centric annotator
Variant-centric annotator
Variant-centric annotator
Variant-centric annotator

![Variant-centric annotator interface](image)

- **Similar fragments**: Similarity: 0.90 --> 0.91
- **Pattern**: medianTest in
- **Variant**: nonmonadic
- **Path**: src/Main.hs

Options available:
- Maintain Equality
- Restore Equality
- Establish Equality (manual)
- Maintain Similarity (manual)
- Increase Similarity (manual)
- Ignore (auto)
Annotation Inference
Similarity Graph

- We can think of the extracted similarities as a graph
  - Fragments are nodes
  - Similarities are weighted edges connecting fragments
- We can use some properties of the graph to infer annotations
  - Mass-annotate equality classes
  - Infer annotations via rules
Equality classes

- An equality class is a set of pairwise equal fragments ("cloning class")

- In the similarity graph an equality class:
  - Complete subgraph with all edges weighted 1.0
  - For n fragments there are $\sum_{i=0}^{n-1} i$ equalities in the class
  - Let the user annotate all equalities with one action!
Equality classes
Annotation inference

\( f_a \rightarrow f_b \rightarrow f_c \rightarrow \)

(Diverge from Equal, -)

(Always Equal, Maintain Equality)

(Diverge from Equal, Restore Equality)
Annotation inference

- We can infer the third annotation
- We can generally write down inference rules
- E.g.:
  - \( A(\{\text{AlwaysEqual, ConvergeToEqual}\}, \text{MaintainEquality, None}) \),
  - \( A(\{\text{DivergeFromEqual}\}, \text{RestoreEquality, Left}) \)
  - \( A(\{\text{DivergeFromEqual}\}, \text{RestoreEquality, Right}) \)
Rule application

- $A(\{\text{AlwaysEqual}, \text{ConvergeToEqual}\}, \text{MaintainEquality}, \text{None})$,
- $A(\{\text{DivergeFromEqual}\}, \text{RestoreEquality}, \text{Left})$
- $\rightarrow A(\{\text{DivergeFromEqual}\}, \text{RestoreEquality}, \text{Right})$
Inference rules

- $A(\{\text{AlwaysEqual, ConvergeToEqual}\}, \text{MaintainEquality, None}),$
- $A(\{\text{AlwaysEqual, ConvergeToEqual}\}, \text{MaintainEquality, None})$
  $\rightarrow A(\{\text{AlwaysEqual, ConvergeToEqual}\}, \text{MaintainEquality, None})$
- $A(\{\text{AlwaysEqual, ConvergeToEqual}\}, \text{MaintainEquality, None}),$
  $A(\{\text{DivergeFromEqual}\}, \text{RestoreEquality, Right})$
  $\rightarrow A(\{\text{DivergeFromEqual}\}, \text{RestoreEquality, Left})$
Cascading rule application

(Always Equal, Maintain Equality)

(Diverge from Equal, Restore Equality)

(Always Equal, Maintain Equality)
Cascading rule application

(Always Equal, Maintain Equality)

(Diverge from Equal, Restore Equality)

(Always Equal, Maintain Equality)

(Diverge from Equal, Restore Equality)
Cascading rule application

(Always Equal, Maintain Equality)

(Diverge from Equal, Restore Equality)

(Diverge from Equal, Restore Equality)

(Always Equal, Maintain Equality)

(Always Equal, Maintain Equality)
Automatic Change Propagation
Automatic change propagation

- For annotations of category Maintain Equality we can easily detect an invariant violation and possibly turn the annotation into Restore Equality.

- For all annotations for category Restore Equality we can perform the expressed maintenance task automatically via change propagation.
Automatic change propagation

- For each **similarity** annotated with Restore Equality
  - If the annotation has a **direction**, overwrite the other fragment
  - Else compute a **three-way-merge** of both fragments
  - If a **conflict** occurs, fail
  - Otherwise, if only **one fragment** differs from the merge content, overwrite other with the merge content
  - Otherwise ask the **user** whether both fragments should be changed
Automatic change propagation

- Changes may propagate over many similarity edges
- Repeat process for all adjacent similarity edges until no change was made in an iteration
Updating annotations

Automatic execution can fail for two reasons

- Three-way-merge has conflicts
- User has declined changing both fragments

Depending on successful or failed executing we need to update the annotation
Automatic change propagation

- If execution failed
  - Set manual flag

- If execution succeeded
  - Update to category *Maintain Equality*
  - Remove direction
Recommending and Updating Annotations
Updating & recommending annotations

- Extract possibly new similarities from new commits
Updating & recommending annotations

- **Recommend invariants** for unannotated similarities
  - Maintain Equality, if fragments are currently equal
  - Maintain Similarity otherwise
Updating & recommending annotations

- **Turn invariants into tasks**
  - Maintain Equality > Restore Equality, if inequality at HEAD
  - Maintain Similarity > Restore Similarity, if similarity value below value when invariant was created
Updating & recommending annotations

- **Turn tasks into invariants**
  - Restore Equality > Maintain Equality
  - Establish Equality > Maintain Equality
  - Increase Similarity > Maintain Equality, if equality at HEAD
  - Restore Similarity > Maintain Similarity, if diff ratio at least as high as when invariant was created
Updating & recommending annotations

- **Recommend** invariants for similarities annotated with tasks
  - Increase Similarity > Maintain Similarity
  - Restore Equality > Maintain Similarity if similarity value is above the one when task was added
  - Remove Equality > Maintain Similarity, if inequality at HEAD
Manual Maintenance
User tasks

- A user might have to act on an annotation if
  - Automatic change propagation failed or
  - The annotation expresses as user task by default
- Ann collects these user tasks in a TODO list
TODO list

<table>
<thead>
<tr>
<th>Name</th>
<th>Auto?</th>
<th>Via rule</th>
<th>Via eq-class?</th>
<th>Intent</th>
<th>Fragment 1</th>
<th>Fragment 2</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase Similarity</td>
<td>Manual</td>
<td>No</td>
<td>No</td>
<td>data/Employee in</td>
<td>data/Employee in</td>
<td>1 → 0.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variant: wxHaskell</td>
<td>Variant: happpstack</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Path: src/Company/Company.hs</td>
<td>Path: Company.hs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restore Equality</td>
<td>Failed</td>
<td>No</td>
<td>No</td>
<td>data/Department in</td>
<td>data/Department in</td>
<td>1 → 0.99</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variant: happpstack</td>
<td>Variant: tmvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Path: Company.hs</td>
<td>Path: Company.hs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Establish Equality</td>
<td>Manual</td>
<td>No</td>
<td>No</td>
<td>pattern/company in</td>
<td>pattern/company in</td>
<td>0.95 → 0.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variant: happpstack</td>
<td>Variant: tmvar</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Path: SampleCompany.hs</td>
<td>Path: SampleCompany.hs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TODO list

- For each annotation the developer has several options
  - Edit fragments to complete manual task
  - Accept/reject a recommendation
  - Remove the annotation
  - Edit the annotation
  - Add annotation to newly extracted similarity
Git Integration
Git integration

- In general many version control systems can be supported
- We have realized the integration with Git
  - Use Git API for metadata extraction
  - Extend set of Git command with new commands
New Git commands

- `git ann init`

  **Functionality:**
  - Registers the repo with our system
  - Triggers initial metadata extraction

  **Integration points:**
  - Any point were repo is in a clean state
New Git commands

- `git ann update`

**Functionality:**
- Updates metadata for new commits
- Updates annotations based on user actions

**Integration points:**
- After new commits are pulled or created locally
New Git commands

- `git ann propagate`

**Functionality:**
- Calls `git ann update` if needed
- Trigger automatic change propagation

**Integration points:**
- Before changes are pushed
Case Study: 101haskell
101companies & 101haskell

- *101companies* documents software languages, technologies and concepts

- *101companies* is a software chrestomathy
  - A collection of small software systems useful for teaching software engineering

- *101companies* is a set of variants all implementing a common feature model

- Variance by
  - (a) implementing different features
  - (b) common features differently to demonstrate languages, techs, concepts
101companies & 101haskell

- 101haskell is a subproject of 101companies
- 101haskell focuses on demonstrating
  - concepts in Haskell
  - technologies for Haskell
- 36 variants (“contributions”)
  - haskellStarter: Contribution with small language footprint
  - haskellComposition: Use of recursive data types
  - haskellVariation: Use of multiple constructors per type
  - wxHaskell: GUI programming in Haskell with wxHaskell
Research question

How to measure the similarity improvements achieved by applying our approach and tools?
Step-wise case study

Four steps and tactics to improve quality of \textit{101haskell}:

1. Automatically restore equalities:
   - Annotate all \textit{equalities}
   - Annotate all \textit{divergences from equalities}
   - Execute \textit{automatic change propagation}
Step-wise case study

- Four steps and tactics to improve quality of 101haskell:

  1. Automatically restore equalities, results:

  - **Fragments**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( \Delta_{0,1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>925</td>
<td>+/-0</td>
</tr>
<tr>
<td>unique</td>
<td>617</td>
<td>-15</td>
</tr>
<tr>
<td>variants sharing (median)</td>
<td>1</td>
<td>+/-0</td>
</tr>
<tr>
<td>variants sharing (average)</td>
<td>1.51</td>
<td>+0.05</td>
</tr>
</tbody>
</table>
Step-wise case study

Four steps and tactics to improve quality of \textit{101haskell}:

1. Automatically restore equalities, results:

\begin{itemize}
  \item \textbf{Similarities}
\end{itemize}

\begin{center}
\begin{tabular}{|c|c|c|}
  \hline
  \textbf{ }} & \textbf{Δ0,1} \\
  \hline
  median & 1.0 & +1.163\% \\
  \hline
  average & 0.96495 & +2.452\% \\
  \hline
\end{tabular}
\end{center}
Step-wise case study

Four steps and tactics to improve quality of 101haskell:

1. Automatically restore equalities, results:

   • Variants, uniqueness (ratio of non-shared fragments)

<table>
<thead>
<tr>
<th></th>
<th>Δ0,1</th>
</tr>
</thead>
<tbody>
<tr>
<td>median</td>
<td>49.00%</td>
</tr>
<tr>
<td>average</td>
<td>51.39%</td>
</tr>
</tbody>
</table>
Step-wise case study

Four steps and tactics to improve quality of *101haskell*:

2. Manually establish equalities:
   - Annotate all *remaining similarities*
   - Manually establish equalities by *editing fragments*
Step-wise case study

Four steps and tactics to improve quality of *101haskell*:

2. Manually establish equalities, results:

*Fragments*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Δ1,2</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>925</td>
<td>+/-0</td>
</tr>
<tr>
<td>unique</td>
<td>595</td>
<td>-22</td>
</tr>
<tr>
<td>variants sharing (median)</td>
<td>1</td>
<td>+/-0</td>
</tr>
<tr>
<td>variants sharing (average)</td>
<td>1.55</td>
<td>+0.04</td>
</tr>
</tbody>
</table>
Step-wise case study

Four steps and tactics to improve quality of 101haskell:

2. Manually establish equalities, results:

- **Similarities**

<table>
<thead>
<tr>
<th></th>
<th>( \Delta 1,2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>median</td>
<td>1.0</td>
</tr>
<tr>
<td>average</td>
<td>0.96795</td>
</tr>
</tbody>
</table>
Step-wise case study

Four steps and tactics to improve quality of *101haskell*:

2. Manually establish equalities, results:

- **Variants, uniqueness**

<table>
<thead>
<tr>
<th></th>
<th>Δ1,2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>median</strong></td>
<td>47.34</td>
</tr>
<tr>
<td><strong>average</strong></td>
<td>50.00</td>
</tr>
</tbody>
</table>
Step-wise case study

Four steps and tactics to improve quality of 101haskell:

3. Increase similarities:
   - Increase similarities by editing
Step-wise case study

Four steps and tactics to improve quality of 101haskell:

3. Manually establish equalities, results:

- Fragments

no change
Step-wise case study

Four steps and tactics to improve quality of 101haskell:

3. Manually establish equalities, results:

**Similarities**

<table>
<thead>
<tr>
<th></th>
<th>(\Delta 2,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>median</strong></td>
<td>1.0 +/-0.0%</td>
</tr>
<tr>
<td><strong>average</strong></td>
<td>0.96802 +0.007%</td>
</tr>
</tbody>
</table>
Step-wise case study

- Four steps and tactics to improve quality of 101haskell:
  3. Manually establish equalities, results:
    - Variants, uniqueness

no change
Results

- Similarity (median/average)
  - Median: 98.85% to 100.00%
  - Average: 94.18% to 96.80%
Results

Sharing

#unique fragments

632 to 595 (-5.85%)

#variants a fragment is shared in (median/average)

1/1.46 to 1/1.55
Results

- Non-trivial equality classes
  - #: 95 to 85
  - max: 13 to 20
Results

๏ Variant uniqueness

๏ Median: 52.47% to 47.34% (-5.13%)
๏ Average: 53.34% to 50.00% (-3.34%)
Results

- **Similarity evolutions**
  
  Initially:
  - Always Equal (38.1%)
  - Always Similar (35.0%)
  - Converge to Equal (8.6%)
  - Diverge from Equal (18.2%)

  After restoring/establishing eqs.:
  - Converge to Equal (32.5%)
Results

- Annotations
  - 324 Restore Equality / unintentional divergences identified
  - 0 Remove Equality
  - 130 Establish Equality
  - 421 Ignore Similarity with 0.8 similarity threshold
    - Most interesting similarities detected
Conclusion
Summary

- We have proposed an approach to manage similarities and cloning across variants.
- We have realized a system to analyze, annotate, and maintain similarities.
- We have presented a case study on 101haskell where the approach helped to improve the projects quality.
Future work

- Support for more operators
  - clone variant, clone feature
- May require additional metadata
- May create annotations automatically
Future work

- Guided annotation process
  - Guide user which similarity to annotate next
  - Maximize number of inferred annotations
  - Minimize number of manual annotations
Thanks! Questions?