



Reuse or Rewrite: Combining Textual, Static, and Dynamic Analyses to Assess the Cost of Keeping a System Up-to-date

*Giuliano (Giulio) Antoniol, Jane Huffman Hayes,
Yann-Gaël Guéhéneuc, and Massimiliano Di Penta*



Software Verification and Validation Research Lab

UK University of Kentucky
Laboratory for Advanced Networking



Università
degli Studi
del Sannio

Content

- Problem statement
- ReORe Idea
- ReORe Process
- Technologies
- The Lynx Browser
- Case Study Results
- Conclusions

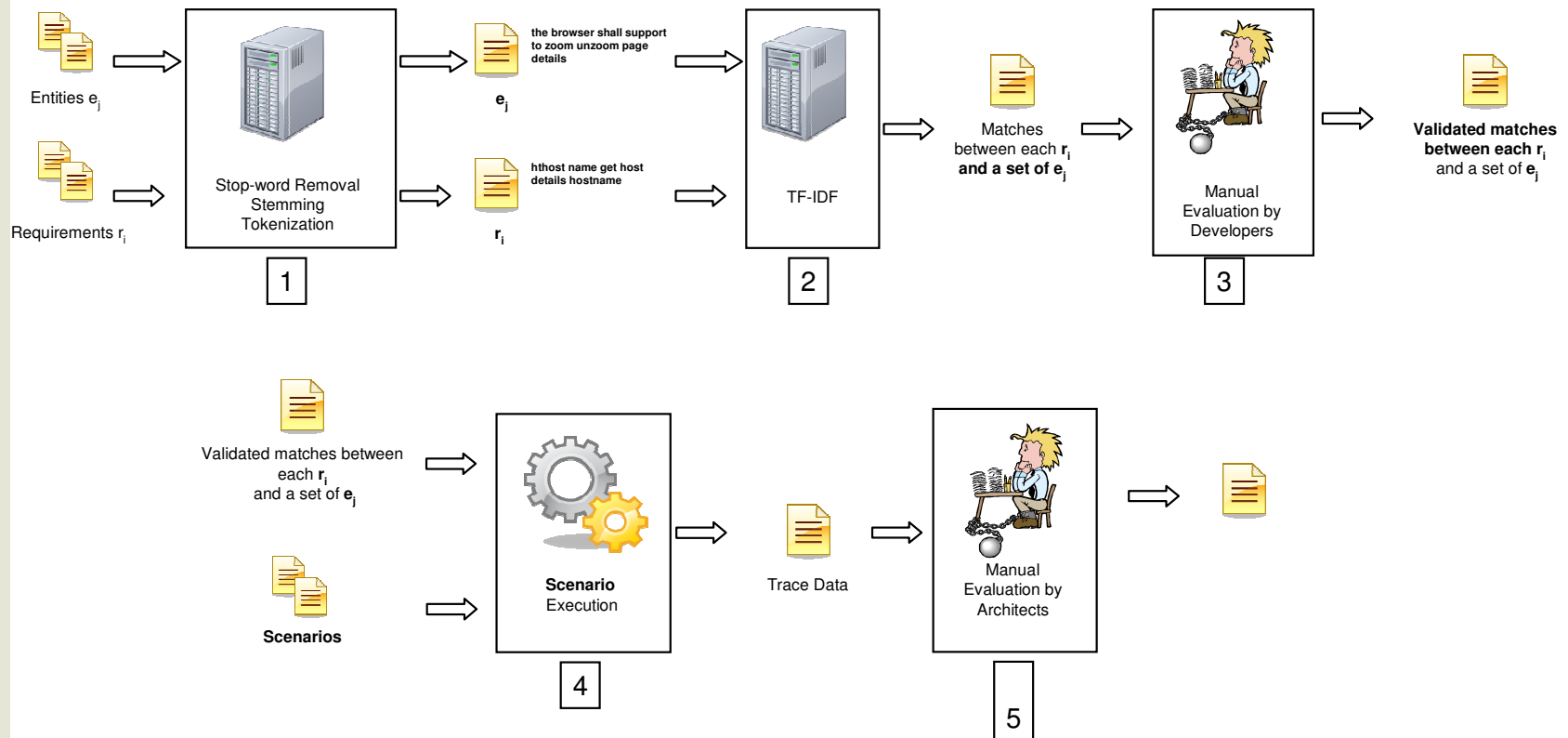
The Challenge

- I own a system S_{prop}
 - it is becoming obsolete, technology, market shifts, etc.;
- My competitors have introduced new technologies, new pieces of functionality that my S_{prop} lacks;
- I can:
 - maintain S_{prop} to bring it up-to-date;
 - rewrite a new system from scratch;
 - get out of the market;
 - Retire and go fishing...

The “Reuse” Or “Rewrite” – ReORe idea

- *Leverage knowledge, existing code, and technologies:*
 - Use competitors systems and your customer base to define a requirement baseline;
 - Assess if it is worth reusing;
 - Apply state of the art technologies to reduce your assessment costs.
- *Resources have different costs:* a junior programmer is not as expensive as the senior architect.
- *Carefully plan the use of resources:* minimize the usage of the most valuable resources.

The ReORe Process



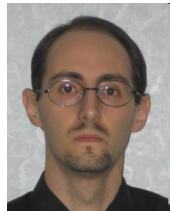
The Company Resources



A young developer applying state of the art technology.



Two (still pretty young 😊) experts programmers filtering the “young” developer’s findings.



The very expensive “guru,” the system architect.

ReORe Technology

- Standard information retrieval vector space model.
- Indexing process:
 - Robust source code parsing and fact extractor;
 - Camel case splitting;
 - Stopper;
 - Stemmer;
 - Thesaurus (not vital but helps);
 - TF-IDF indexing.
- Scenario-based dynamic trace collection and analysis.

PrereqIR (see TR or WCRE'08 paper)

- We need a pre-requirement document:
 - what the competitor systems do;
 - what our customer base needs.
- Obtain and vet a list of requirements from diverse stakeholders.
- Structure the requirements by mapping them into a representation suitable for grouping via pattern-recognition and similarity-based clustering.
- Analyze the clustered requirements to divide them into a set of essential and a set of optional requirements.



Step 1 – Inexpensive resources

- Process both:
 - the set of requirements R ;
 - the set of entities E extracted from the source code.
- Any requirement or entity is mapped into a textual representation.
- Map textual representation into vector space.



Step 2 – Inexpensive resources

- Trace requirements into code entities.
- Apply a threshold:
 - to remove low similarity matches;
 - reduce manual verification.
- Compute the threshold via outlier identification;
 - much likely it is better to get a very tight threshold.
- Overall, try to reduce the search space of orders of magnitude.

Step 3 – Experts/Developers

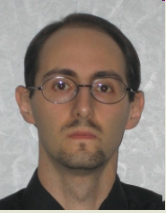
- Manually evaluate the rankings from Step 1 – 2.
- For each requirement, only retain entities believed to concretely participate in the implementation.
- Tag selected entities as reusable.
- This phase is necessarily manual:
 - only the developers can assess, using their knowledge of the system and contextual information, whether an entity *really* participates in the implementation of requirement.





Step 4 – Inexpensive and patient...

- Build and execute scenarios:
 - exercise the requirements for which a consensus concerning their implementing entities exist.
- Develop two kinds of scenarios:
 - **Core scenarios:** scenarios exercising core functionalities and requirements;
 - **Specific scenarios:** scenarios specifically devoted to some particular requirement.
- Use core scenarios to identify the core implementation entities:
 - the infrastructure of the system.
- Validate if entities mapped in Step 1 – 3 really show up.



Step 5 – Most valuable resource

- Identifies and validates the entities implementing a requirement.
- Works on the reduced information produced in Step 1 – 4.
- While validating data, can also recover missing links and entities.

Case Study

Research questions:

- **RQ1:** is ReORe able to effectively reduce the information processed in the different steps while ensuring the traceability between requirements and the existing code?
- **RQ2:** what is the effort required for the application of ReORe?

Context:

- Reuse components from a text-oriented Web browser (Lynx) to develop a modern browser (e.g., Firefox)
- Lynx is a text oriented (curses) browser developed in C
 - 91 source files about 147 KLOC and 2074 functions;
 - 156 header and configuration files about 27 KLOC;
 - some nice macros...
- Requirements:
 - 128 essential;
 - 181 optional.

Threshold Determination

- Out of $128+181 = 309 \times 2,074 = 640,866$ links, 138,896 have similarity > 0 .
- Similarity:
 - median 0.012;
 - percentiles 25%: 0.006 – 75%: 0.026 - IQR: 0.021.
- Standard outlier threshold 6% ($0.026 + 3 \times 0.021$) still too high – gives about 12,000 matches...



- Not really happy...
Take $8/9$ times IQR which gives 20%.

Requirement Example

“the system shall allow internet cookies to be downloaded in the local file system”

Functions	Ranks
src/LYCookie.c::newCookie	0.3489
src/LYCookie.c::LYProcessSetCookies	0.3088
src/LYCookie.c::LYSetCookie	0.2942
src/LYCookie.c::ARGS1	0.2797
src/LYDownload.c::LYdownload_options	0.2559

Not all requirements have at least one candidate link with similarity greater than 20%



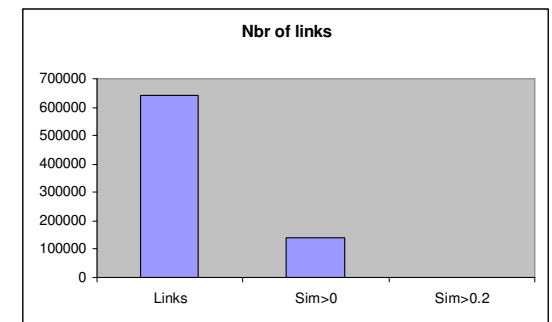
Search Space – First Reduction

Input Step 1: $309 \times 2,074 = 640,866$ Links



138,896 Links with
Similarity > 0

Output Step 3: 738 Links with
Similarity $> 20\%$

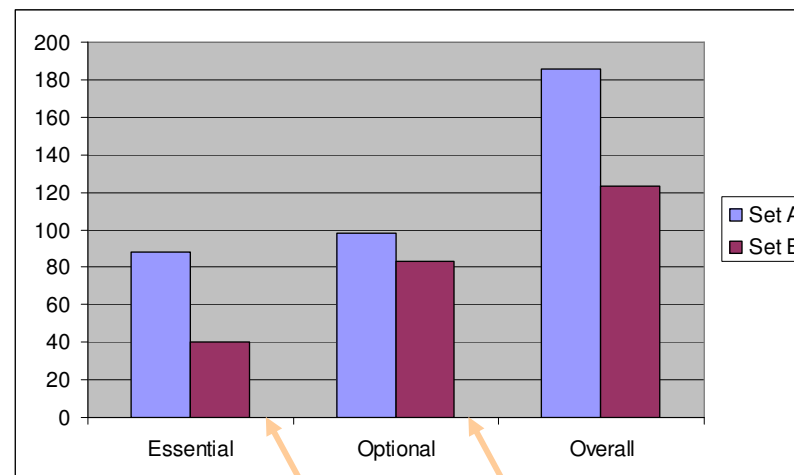




Matching Details

Set A: at least one match > 0.2

Set B: no match above 0.2



40

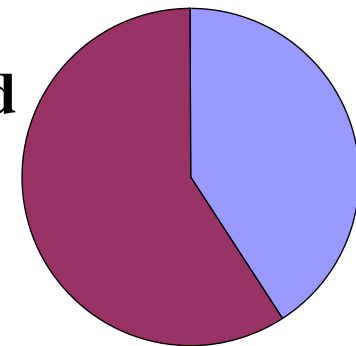
83

A further problem:
Average list length

464 (A) 425 (B)

Essential REQ Mapped

Mapped
68%





The Junior Developer Dilemma

- Reduce to 738 possible links.
- Know that only $88 + 98 = 186$ requirements have at least one link >0.2 , but:
 - each link has an average 450 matches;
 - unsure if the “experts” will vet by hand an average of 450 links...
- Trick the “experts”
 - just take the top 5 matches;
 - add another 5 matches extracted from the list with uniform distribution.
- Perhaps discover missed links, at least make sure experts do not cheat...

Step 3 – Expert Manual Decision

Set	Ranking Range	Yes	No	Only One Yes
A	1-5	128	226	324
	6-10	2	790	20
B	1-5	25	407	69
	6-10	0	570	10

Experts agree more often on top five in set A!
Also agree most of the time for a NO on set B

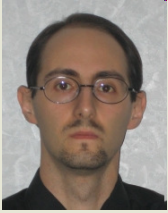
Step 4 – Core functionalities

Any browser will handle HTTP, SSL, and FTP

Scenario	Number of Functions
HTTP	382
SSL	137
FTP	28

Step 4 – Dynamic Analysis Filter

Set	Ranking Range	Retained	Discarded
A	1-5	77	51
	6-10	1	1
B	1-5	10	15
	6-10	-	-



Step 5 – Architect Decision

Set	Ranking Range	Retained	Discarded	Added
A	1-5	73	4	12
	6-10	1	-	-
B	1-5	9	1	1
	6-10	-	-	-
Overall		83		13

3 are duplicate reqt.

Requirement	Set A	Set B
Essential (128)	23	5 (2)
Optional (181)	23	4
Overall	46	9

Overall: 25 essential requirements and 27 optional requirements.

Accuracy

		Accuracy	
		Dynamic Analysis	Final Inspection
Set A	1-5	60%	95%
	6-10	50%	-
Set B	1-5	40%	90%
	6-10	-	-
Overall		56%	94%

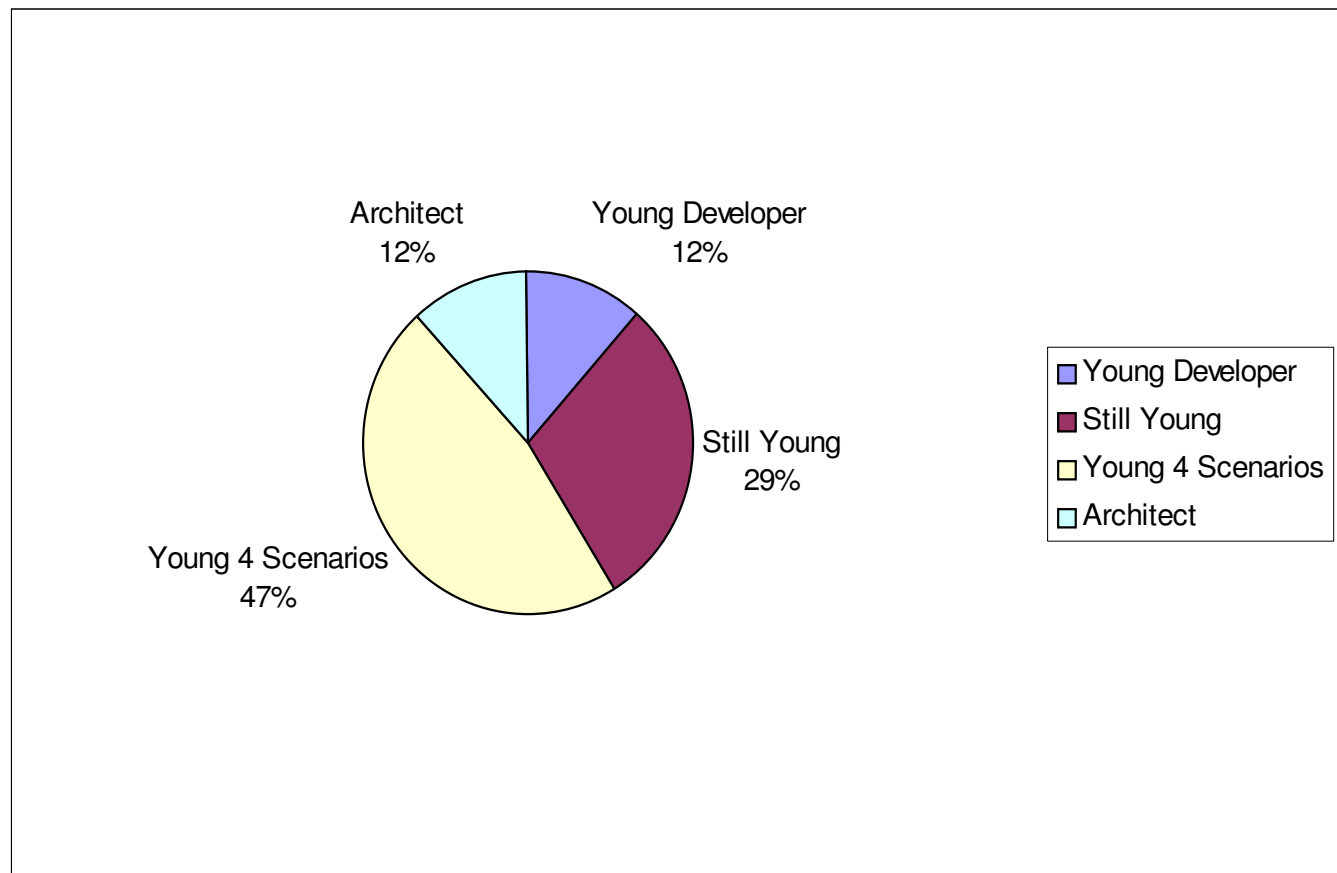
Each step plays the role of the Oracle for previous steps.

Accuracy: the percentage of matches for which an agreement between any two subsequent steps was found.

It is representative of the effort saved.

Effort (RQ2)

Young Developer	10
Still Young	25
Young 4 Scenarios	40
Architect	10



Discussion

- Clearly there is manual activity involved.
- Company knowledge should often be sufficient.
- Some of the functions contribute to core requirements as well as to specific requirements (the switch course).
- Only one function in Set B was not in Set A:
 - don't even try to collect if similarity is low.
- Overall, we spent 85 – 90 hours of work.

Threats to validity

- **External validity (generalization of findings):**
 - Particular case study adopted → results may or may not be similar for other systems;
 - Participants were from academia;
- **Construct validity (relationship theory-observation):**
 - Thresholds play a big role in ranking requirements;
 - Experimented different thresholds;
 - Dynamic analysis depends upon test cases;
 - Final validation aims at reducing construct validity threats;
- **Internal validity (external factors might influence results):**
 - Multiple steps with different techniques reduce the degree of subjectivity;
 - Architect involved in the last step was not aware of how the previous steps were performed;
- **Reliability validity (repeatability of results):**
 - Process fully detailed;
 - System available;
 - Other data available upon request.

Conclusions

- ReORe uses standard state of the art technologies:
 - Eases the task of deciding “**Reuse**” Or “**Rewrite**”.
- ReORe effectively filters information reducing the manual intervention of most expensive resources.
- For our case study (Lynx):
 - it doesn't pay-off investigating low similarity matches;
 - only 25 % of code can be reused and only 25 essential requirements out of 128 have some initial implementation.

The longer you wait, the worst it gets

Questions ?