SYSTEMS ENGINEERING
Research Center

SMU

Conceptual Framework of Managing Technical Debt

Qiao Zhang
Advisor: LiGuo Huang

Dept. of Computer Science and Engineering
Southern Methodist University
{qiaoz, lghuang}@smu.edu

PSM Users' Group Workshop

What is Technical Debt?

Technical Debt (TD): Delayed technical work or rework that is incurred when shortcuts are taken or short-term needs are given precedence over the long-term objectives. It is the result of intentional or unintentional decisions that impact the viability of a system and usually incur interests (i.e., additional cost) to eliminate.

- *Process/Methods*
- *Schedule*
- *Capabilities*
- *Budget*
- *Documentation*
- *…*

What is Technical Debt Item/List?

Technical Debt Item (TDI)*: Single atomic occurrences of TD in a project.

- *Suboptimal design*
- *Problematic source code*
- *Skipped test case*
- *…*

Technical Debt List (TDL)*: A list to maintain all TDI information in a project.

- *TDI type and description*
- *Discovered time and location*
- *Estimated principal and interest*
- *…*

* Holvitie, J., & Leppanen, V. (2013). DebtFlag: Technical debt management with a development environment integrated tool. *2013 4th International Workshop on Managing Technical Debt, MTD 2013 - Proceedings*, 20–27. http://doi.org/10.1109/MTD.2013.6608674

What is TD Principal and Interest?

TD Principal: Estimated costs/efforts of paying off the technical debt item (TDI) when it is detected.

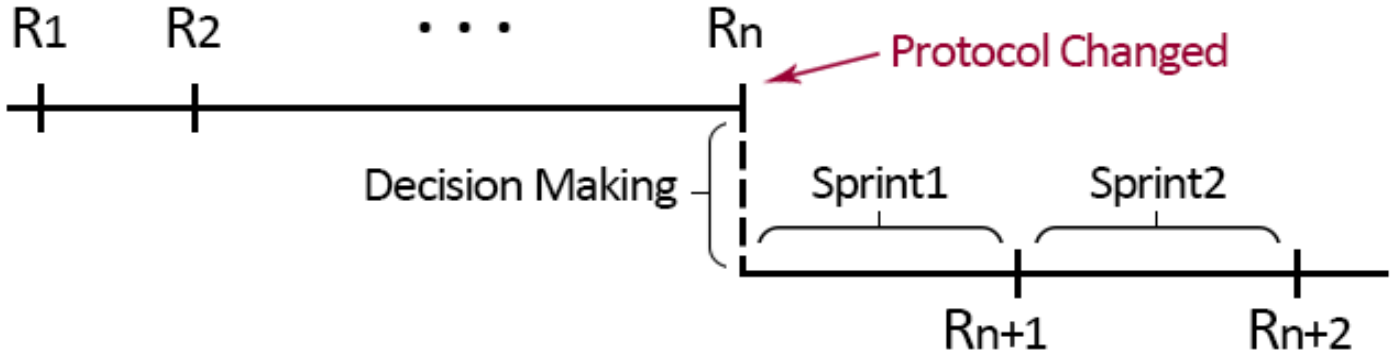
TD Interest: Continuing costs/efforts due to the delay of treatment of TDI after it is detected. (e.g., higher maintenance costs, greater resource usage, etc.)

- *The relationship between TD interest and time is not linear*
- *There might be zero TD interest (or pay the interest)*
- *TD interest can be measured by calculating the distance between two estimation of TD principal at different time*

$$TD\ interest = \delta = p_t - p_{t+X}$$

Where, t is the time when we last estimate the principal of the TDI, and X is the interval we choose to update the TDL iteratively.

An Exploratory Example: TD Interest Amount and Interest Probability



* *ABC* is a mobile app as a client of server system *E*, which is developed by team *M*. In the previous releases (R_1-R_n), all features were built upon the protocol *A* for communication with *E*. During preparation of release R_{n+1}, team *M* declares that the protocol *A* needs to be replaced with protocol *B* since it is no longer supported by the new server system *E'*. As a result, app *ABC* may need to support both protocol *A* and *B* in the near future.

- **TD Principal (P):** $\forall M_k \in R_n, P = \sum_k D(M_k, A) \times C_r(M_k)$
- **TD Interest Amount:** $\forall M_j \in R_{n+1}, P = \sum_j D(M_j, A) \times C_i(M_j)$
- **TD Interest Probability:** probability that system *E* is upgraded to *E'* by customers

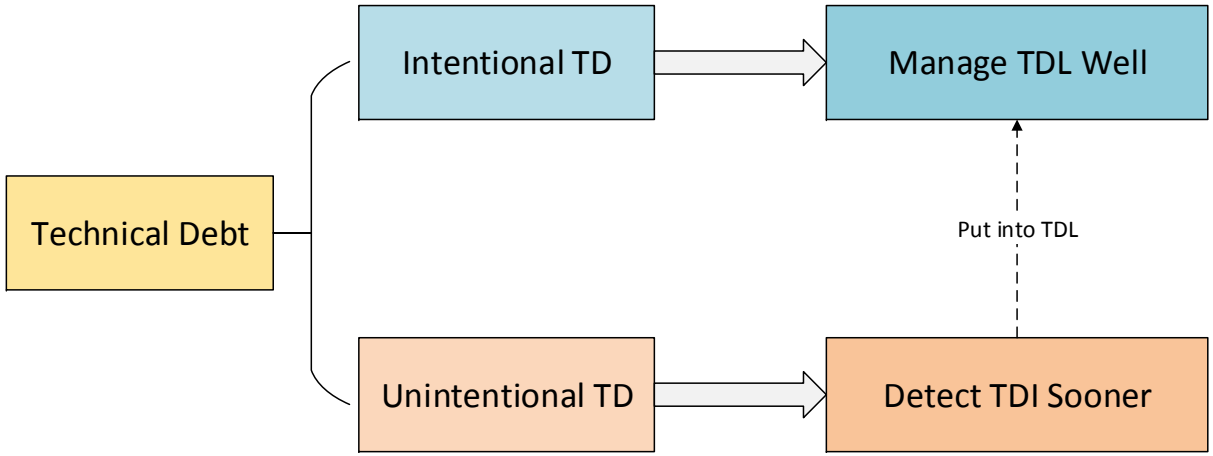
M_k: module k. | $D(M_k, A)$: dependency between module *k* and protocol *A;*
$C_r(M_k)$: rework cost of module *k ;*
$C_i(M_j)$: development cost of new feature module *j .*

* Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. Da Silva, A. L. M. Santos, C. Siebra, Tracking technical debt – an exploratory case study, in: Software Maintenance (ICSM), 2011 27th IEEE International Conference on, IEEE, 2011, pp. 528 – 531.

Theories of TD Management

- Not all TD is bad. ➔ Some intentional TD is able to create a lot of return on investment (ROI) if managed well.
- Don't expect people to admit their flaws willingly. ➔ The sooner we detect unintentional TD, the better we can manage them.

For better management, we need update TDL constantly with an appropriate interval (e.g., a spiral, a sprint, or a month, etc.) for tracking the information of important TDIs.

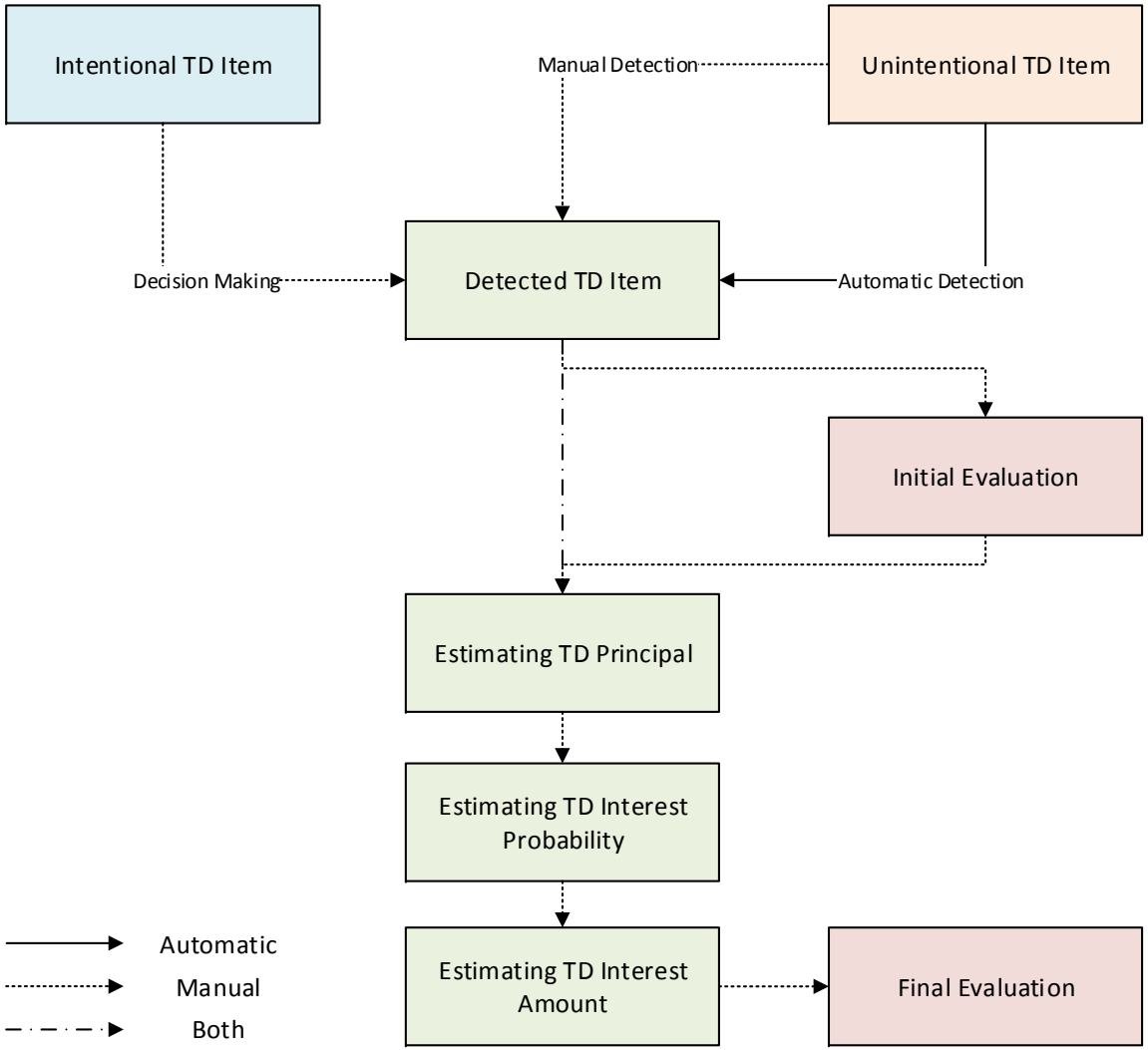


Choose TDI with TD Categories

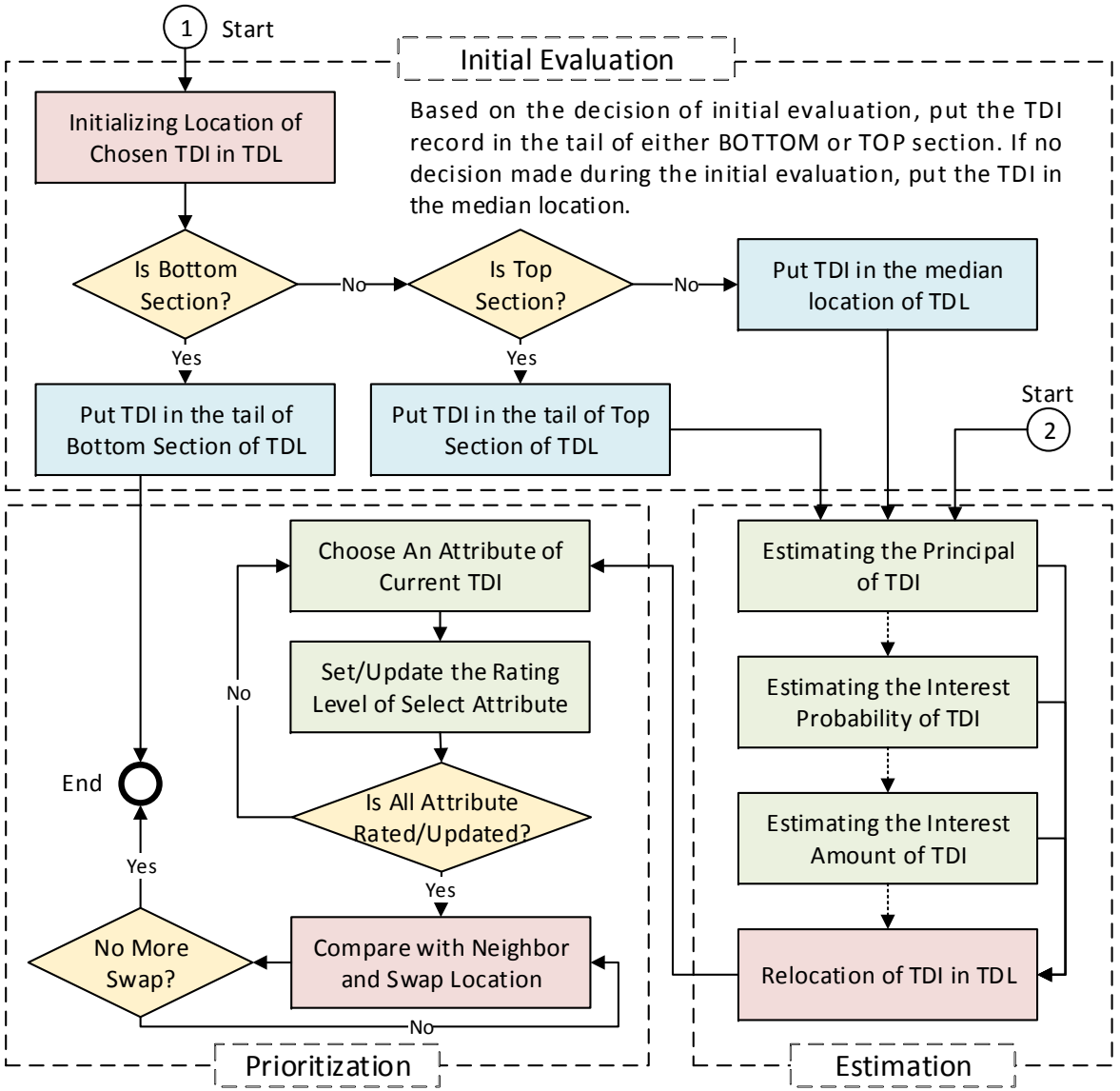
TD Categories	Definition	TDI
Requirement Debt	The distance between the optimal solution and the actual solution with respect to some decision space	Sub-optimal decisions
Architecture Debt	Degraded architecture quality or sub-optimal solution	Architecture drifts
		Architecture erosions
Design Debt	Violation of the principles of good object-oriented design	Code smells
Code Debt	Source code that negatively affect the legibility of the code	
Maintenance Debt	Delayed certain software maintenance tasks	Unfixed bugs
		Latent defects
Test Debt	Lack of test scripts or insufficient test coverage	Skipped test cases

...

Detection, Estimation, and Evaluation of TDI



Prioritizing TD Items in TD List



Example TD List and Final Decision

TDI #	TDI Type	Est. Principal	Est. IP	Est. IA	Location	Personnel	Resource Consumption
1	Architecture drift	5K	85%	2K	5	3	3
2	God Class	3K	85%	1K	4	2	4
3	Data Class	3K	80%	2K	3	5	2
4	God Class	1K	75%	--	4	2	2
5	Design Flaw	4K	90%	--	2	1	3
…	…	…	…	…	…	…	…
20	Unfixed Bug	2K	45%	1K	--	--	--
21	Duplicate Code	2k	--	--	--	--	--
22	Latent Defect	2K	70%	1K	3	1	2
…	…	…	…	…	…	…	…
51	Duplicate Code	--	--	--	--	--	--
52	To-do Tag	--	--	--	--	--	--

…

To pay off the TDI, choose from: (1) refactor, (2) replace, (3) block, and (4) abandon

Refactoring the TDI with various refactoring operations is the most common way to perform the payoff of the TDI.

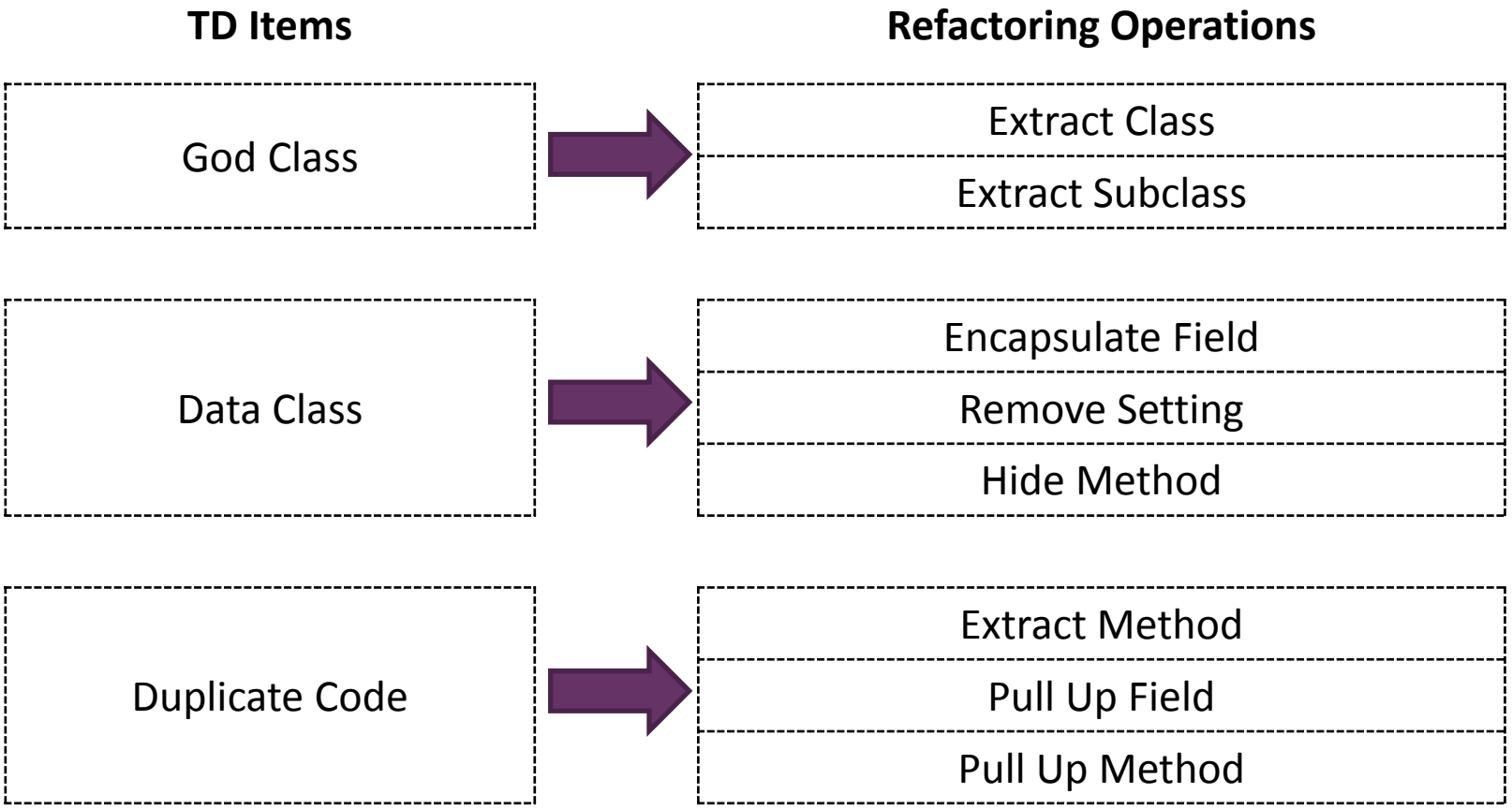
The percentage of design changes due to the refactoring operations on TD items

$$AAF = (0.4 \times DM) + (0.3 \times CM) + (0.3 \times IM)$$

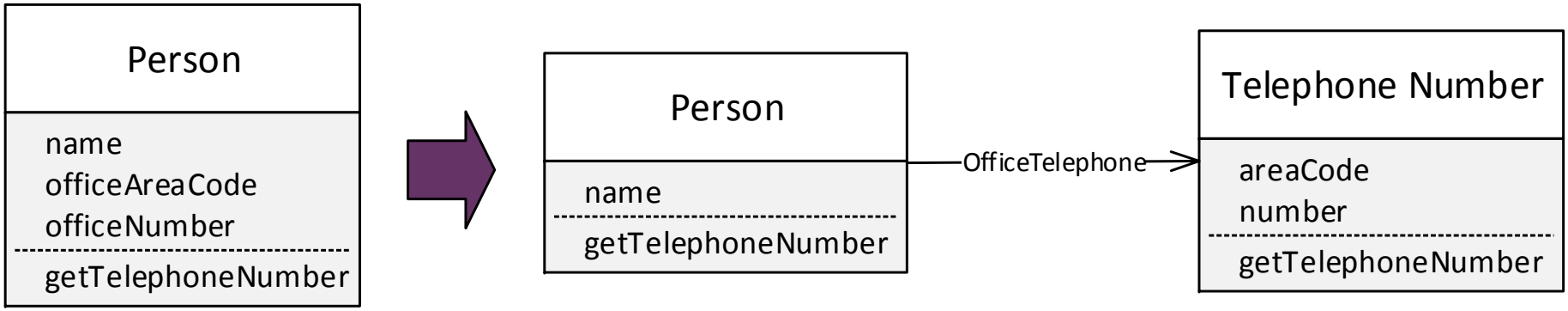
The percentage of LOC changes due to the refactoring operations on TD items

$$PM_M = A \times Size_M{}^{E} \times \prod_{i=1}^{15} EM_i$$

TD Items and Refactoring Operations



Refactoring God Class: Sizing Change for Extracting Class Operation



- **Prediction Function*:** Extract from a source class c_s with a set of attributes $\{a_k\}$ (where $n = |\{a_k\}|$) and a set of methods $\{m_h\}$, to a target class c_t. While $l = 6$ when program language is JAVA.

$$LOC_p(c_s) = LOC_b(c_s) - (nl\ - n + \sum_h LOC(m_h))$$

$$LOC_p(c_t) = nl + \sum_h LOC(m_h) + p$$

* Chaparro, O., Bavota, G., Marcus, A., & Di Penta, M. (2014, September). On the impact of refactoring operations on code quality metrics. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*(pp. 456-460). IEEE.

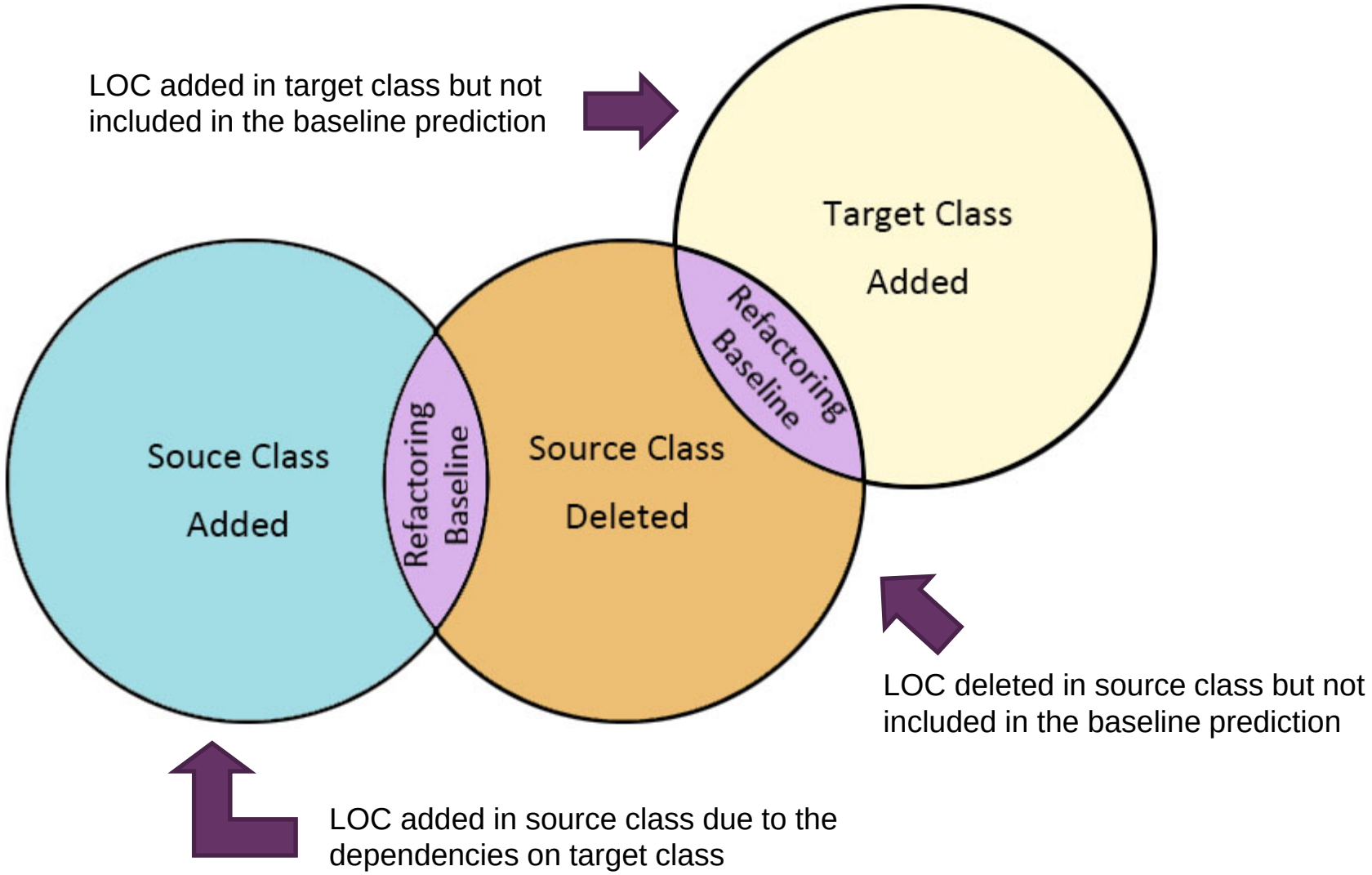
Sizing Change Prediction Model for Refactoring Operations

Refactoring Operation	Prediction Function*	Required Input
Move Method	$LOC_p(c_s) = LOC_b(c_s) - LOC(m_k)$ $LOC_p(c_t) = LOC_b(c_t) + LOC(m_k)$	Identify source class c_s, target class c_t, and method m_k to be moved
Extract Method	$LOC_p(c_s) = LOC_b(c_s) + l + 1$	Set the value of l based on programming language
Inline Method	$LOC_p(c_s) = LOC_b(c_s) + l + 1$	Set the value of l based on programming language
Pull Up Field	$LOC_p(c_s) = LOC_b(c_s) - 1$ $LOC_p(c_t) = LOC_b(c_t) + 1$	Identify subclass c_s and super class c_t
Pull Up Method	$LOC_p(c_s) = LOC_b(c_s) - LOC(m_k) + l(d_k + p_k)$ $LOC_p(c_t) = LOC_b(c_t) + LOC(m_k) + l_k + d_k + p_k)$	Identify subclass $c_s$super class c_t, method m_k to be moved, d_k (number of distinct fields being read in m_k), and p_k (distinct fields being written in m_k)
Replace Inheritance with Delegation	$LOC_p(c_s) = LOC_b(c_s) + 1$ $LOC_p(c_t) = LOC_b(c_t)$	Identify source class c_s, target class c_t,

…

* Chaparro, O., Bavota, G., Marcus, A., & Di Penta, M. (2014, September). On the impact of refactoring operations on code quality metrics. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*(pp. 456-460). IEEE.

Investigation of Additional Factors for Extract Class/Subclass Operation



Example: Comparing LOC Deleted in Source Class with Target Class

Base Text		New Text	
		1	/*
		2	* Licensed to the Apache Software Foundation (ASF) under one or more
		3	* contributor license agreements. The ASF licenses this file to You
		4	* under the Apache License, Version 2.0 (the "License"); you may not
		5	* use this file except in compliance with the License.
		6	* You may obtain a copy of the License at
		7	*
		8	* http://www.apache.org/licenses/LICENSE-2.0
		9	*
		10	* Unless required by applicable law or agreed to in writing, software
		11	* distributed under the License is distributed on an "AS IS" BASIS,
		12	* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
		13	* See the License for the specific language governing permissions and
		14	* limitations under the License. For additional information regarding
		15	* copyright in this work, please see the NOTICE file in the top level
		16	* directory of this distribution.
		17	*/

Example (Cont.): Additional Size Change Due to Refactoring Operation


```
113
114 @Override
115 protected ResponseContext process(
116 Provider provider,
117 RequestContext request) {
118 TargetType type = request.getTarget().getType();
119 TypeHandler handler = getHandler(type);
120 if (handler == null) {
121 String method = request.getMethod();
122 handler = getMethodHandler(type,method);
123 if (handler == null) {
124 return super.process(provider, request);
125 } else {
126 return handler.process(provider, request);
127 }
128 } else {
129 return handler.process(provider, request);
130 }
131 }
132
133 }
134
```

Conclusions

- Technical Debt (TD) can be measured, estimated, and evaluated as Technical Debt Items (TDI).
- The prioritization of TDIs in the Technical Debt List (TDL) is to support the decision making during the TD management.
- The development/maintenance effort due to TDIs is promising to be estimated with COCOMO by reuse model and maintenance model.
- Predicting sizing change due to TD item refactoring can be an automatic approach, which saves a lot of effort.

Thank you!

Qiao Zhang
Advisor: LiGuo Huang

Dept. of Computer Science and Engineering
Southern Methodist University
{qiaoz, lghuang}@smu.edu

PSM Users' Group Workshop