Q2.2: What tools do you currently use for refactoring at any scale?	
Q4.10: What tools, if any, did you use to assist your large-scale refactoring efforts?	
Code	**Description**
Code smells analysis tools	Tools that use static analysis to identify areas of the code that do not conform to accepted implementation rules and design best practices
Continuous integration	Tools used to integrate code changes from multiple contributors
Dependency exploration	Tools that enumerate dependencies in code and their properties
IDE	General reference to integrated development environments without specifying the purpose of use
IDE refactoring features	Specific reference to refactoring functionalities of integrated development environments
Manual efforts	Using techniques that do not apply automated support
Other	Responses that do not fit one of the remaining categories
Refactoring tool	Tools that recommend refactorings and realize their implementations
Testing tools	Tools that assist with unit, regression and integration testing
Text editors	Tools that are used to edit plain text, in our context those basic ones primarily used for editing code
Version control/Issue tracker	Tools used for change management, list of issues in the software
Visual modeling	Tools used to visually represent software, problem analysis, decision making; could apply to before or after refactoring

Q4.2: What were the business goals of the refactoring?	
Q5.2: For what business reasons did you want to perform a large-scale refactoring?	
Code	**Description**
Capability	To add or improve or facilitate the improvement of the external (i.e., user-facing) capability. This encompasses features and functionality
Cost	Reduce the total cost of ownership. This includes the cost of: development, deployment and operations (license fees), maintenance, and moving to a new system
Market position	Expand or retain market share, maintain or improve reputation of business (customer service, influence hiring), reputation of product, enter new markets, reduce time to market
Modernization	To remove reliance on older / end-of-life technology, migrating to a new system, changing programming language
Productivity	Support improved business processes such as agile practices, distributed development, follow modern code practices. Improve software development productivity (e.g., team motivation, increase velocity, follow modern code practices)
Quality	To improve the external (i.e., user-facing) quality of the system. This encompasses defects (count and rate), performance and scaling, usability, and security

Q4.3: What were the technical goals of the refactoring?	
Q5.3: For what technical reasons did you want to perform a large-scale refactoring?	
Code	**Description**
Capability	To add or improve or facilitate the improvement of the external (i.e., user-facing) capability. This encompasses features and functionality
Cost	Reduce the total cost of ownership. This includes the cost of: development, deployment and operations (license fees), maintenance, and moving to a new system
Maintainability	To improve the internal quality of the code
Modernization	To remove reliance on older / end-of-life technology, migrating to a new system, changing programming language
Productivity	Support improved business processes such as agile practices, distributed development, follow modern code practices. Improve software development productivity (e.g., team motivation, increase velocity, follow modern code practices)
Quality	To improve the external (i.e., user-facing) quality of the system. This encompasses defects (count and rate), performance and scaling, usability, and security
Reuse	A goal of the refactoring was to facilitate greater reuse (i.e., common services, software product line)
System resources	Improve utilization of software and hardware platform

Q4.5: What other significant activities did you perform during refactoring?	
Code	**Description**
Communication	Communicating and building trust with management, customers, and other developers. E.g., getting developers to buy into the goals of the refactoring, getting funding and other resources, etc.
Comprehension	Understanding existing requirements, assumptions, and constraints
Education	This covers both explicitly training/onboarding developers and writing documentation (e.g., explaining structure, rationale, and decisions)
Evaluation	Verifying, validating, and certifying the changes. Assessing the consequences of the refactoring on the rest of the system and ensuring backwards compatibility
Operations	Support operational and deployment activities and stakeholders (e.g., migrating and managing users).
PlanningHow	Deciding exactly how the code should be refactored. E.g., given what to refactor as input (e.g., design problems), produce several refactoring options
Process	Understanding how refactoring fits into the larger software development process. E.g., performing agile test-driven development (TDD), managing version control and parallel development, etc
Use of tools	Develop, acquire, and use tools to help in the refactoring (whether custom-made or off-the-shelf)

Q4.8: For the most-challenging activities that you identified, what made those activities challenging?	
Code	**Description**
Lack of code comprehension	Difficulty understanding existing code due to scale, dependencies, or side effects
Lack of code quality	Poor code quality
Lack of communication	Gaining stakeholder cooperation (including management support), managing expectations, and gaining user trust
Lack of decision criteria	Deciding whether one candidate refactoring is better than another depends on the priorities of competing goals. Without guiding metrics, it can be difficult to evaluate and compare candidate refactorings
Lack of documentation	A lack of documentation and unclear requirements for the original system
Lack of refactoring techniques	A lack of well-defined refactoring techniques make large-scale changes difficult and ad-hoc
Lack of tests	Without tests, it is difficult to assess the correctness of changes
Scoping the refactoring	Deciding what changes should be considered in scope for the refactoring and assessing whether those changes are worth making

Q4.11: What kind of automation, if available, would have most improved your large-scale refactoring?	
Code	**Description**
Analysis	Improved static and dynamic analyses, ability to assess changes
Build automation	An ability to automate the process of rebuilding the codebase following changes (i.e., continuous integration)
Comprehension	Tooling that would generate abstract views of code (e.g., architecture), improved visualization
Modification	Tooling to automatically and safely apply pervasive changes to the codebase with minimal user input
PlanningHow	Tooling that, given what to refactor as input, produces and compares potential refactoring options
PlanningWhat	Tooling that helps to identify meaningful design problems and opportunities for refactoring
Testing	Generate unit tests (prior to refactoring), refactor tests in parallel with refactored code, automate other forms of testing (integration, regression, acceptance, application, and review)

Q5.4: For what reasons did your organization decide not to perform the large-scale refactoring?	
Code	**Description**
Comprehension	It can be difficult to understand how the existing operates and how/where refactoring should take place in that system to achieve the desired goals. Poor comprehension may come from a lack of tests, documentation, requirements, or simply poor internal code quality.
Management	A lack of support or funding from leadership/management (e.g., due to a general organizational inertia)
Staffing	Insufficient, undertrained, or otherwise unhappy staff are available to support the refactoring.

Q5.6: What consequences, if any, did you observe from not performing the refactoring?	
Code	**Description**
Delivery	The system was not delivered according to plan (e.g., it took longer than expected, was shipped without certain features, or was not shipped at all)
External quality	The user-facing quality of the product deteriorated due to, e.g., bugs, vulnerabilities, or degraded user experience
Internal quality	The internal quality of the codebase deteriorated, making it harder to work with and thus lowering productivity
Staffing	Adverse effects on development staff (e.g., difficulty hiring; low team morale; increased onboarding time)

Q7.1: What are the strengths and weaknesses of the refactoring tools, if any, that you currently use?	
Code	**Description**
This question inherits all of codes from Q4.11, in addition to the codes below	
LSR (large scale refactoring)	Existing refactoring tools are good at “dumb” and shallow refactoring, but are unable to apply intelligence to the problem to generate deeper refactorings at a large scale
Scoping the refactoring	Existing refactoring tools provide little or no assistance in deciding the scope of a refactoring (e.g., estimating the value of a refactoring; identifying opportunities for reuse; satisfying business goals)
Usability	Properties of the tool and use by developer (cost, code size, skill, selection, setup, alignment with standards, time, transparency, workflow)