Operationalizing Software Engineering Theories for Practical Validation
Appendix-C. Refinement of the meaning of the main indicators

- **autonomy: self-organization**. A team can access to all necessary information to develop, deploy, and operate the product. “Now developers have autonomy for going from zero to production without having to wait for anyone”.
- **autonomy: dependent**. A team depends on another team/party at some point in the product lifecycle.
- **blame**. It is part of the team culture values. It occurs when the Dev or Ops team blames the other for a problem or error, usually occurs when they don’t share the responsibility.
- **alignment of dev&ops goals: product thinking.** Dev and ops goals are aligned towards the product. “A collaborative culture requires product thinking, in substitution to operations or development thinking.”
- **alignment of dev&ops goals: local optimization.** Dev and ops have their own goals.
- **responsibility/ownership sharing: full sharing.** Dev and ops share responsibility of products, output artifacts (e.g., databases), and tasks (e.g., NFR shared responsibility, infrastructure management shared responsibility, monitoring shared responsibility, and incident handling shared responsibility, etc.) so that everyone is responsible for build, test, deploy, operate and maintain their applications and infrastructure.
- **responsibility/ownership sharing: medium sharing**. Dev and ops may share some responsibility of products, output artifacts (e.g., databases), and tasks (e.g., NFR shared responsibility, infrastructure management shared responsibility, monitoring shared responsibility, and incident handling shared responsibility, etc.) although some responsibilities may still reside on one side (dev or ops).
- **responsibility/ownership sharing: null sharing**. Dev and ops have separate responsibilities and tasks (each team member has different responsibilities and tasks).
- **skills/knowledge sharing: full sharing**. High levels of knowledge sharing (e.g., developers may have knowledge about infrastructure/platform).
- **skills/knowledge sharing: medium sharing**. Medium levels of knowledge sharing or minimal awareness of what is happening on the other side of the wall.
- **skills/knowledge sharing: null sharing**. Low levels of knowledge sharing or no awareness of what is happening on the other side of the wall.
- **Stack & tools sharing: full sharing**. A holistic view of the tools and stack——the frontend, backend, libraries, storage, kernels, and physical machine, which have a high degree of standardization and integration.
- **stack & tools sharing: medium sharing**. Partial view of the tools and stack. Not all the tools have been standardized and integrated, such monitoring tools.
- **stack & tools sharing: null sharing**. Non-shared stack.
- **cross-functionality/skills**. An attribute of teams that have the necessary skills to achieve end-to-end ownership.
- **role definition/attributions**. When true, it indicates when a team has well-defined roles and are less likely to share responsibility. When false, it indicates that the teams do not have a well-defined definition of responsibility.
- **change type: small&frequent**. With a small batch size (i.e. the unit of work that passes from one stage to the next stage in a process), each batch makes it through the full lifecycle quicker, at least once a day according to Fowler definition for continuous integration.
- **change type: large&rare**. Large batch size hampers continuous integration and delivery. “Pushing 2 weeks of commits to prod is a larger batch than pushing every commit in a continuous delivery system”.
- **collaboration frequency: eventual/daily**. The frequency of collaboration between teams when they share responsibility for an activity.
- **collaboration quality: high/low**. It represents the quality of collaboration between teams.
- **communication type: poor/rare or frequent**. It presents the level of communication established between the teams, which can range from rare to frequent.