Research Reports in Software Engineering and Management 2012:01 ISSN 1654-4870

FACTORS INFLUENCING REUSE AND SPEED IN THREE ORGANIZATIONS

CHALMERS | UNIVERSITY OF GOTHENBURG

Antonio Martini

Department of Computer Science and Engineering

FACTORS INFLUENCING REUSE AND SPEED IN THREE ORGANIZATIONS

Antonio Martini

© Antonio Martini, 2012

Report no 2012:01
ISSN: 1651-4769
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden

Telephone + 46 (0)31-772 5710

Göteborg, Sweden 2012

Research Reports in Software Engineering and Management No. 2012:01

FACTORS INFLUENCING REUSE AND SPEED IN THREE ORGANIZATIONS

Antonio Martini
antonio.martini@chalmers.se

Department of Computer Science and Engineering
CHALMERS | University of Gothenburg

Gothenburg, Sweden 2012

FACTORS INFLUENCING REUSE AND SPEED IN THREE ORGANIZATIONS

Antonio Martini

Introduction

This report supplements the research paper Martini et al, "Inhibitors and Enablers for Reuse with Speed". It lists main factors influencing reuse and speed in three organizations. Factors come from the analysis of a set of 7 interviews, 3 of which have been used as secondary evidences. The report also includes the interview protocol of the study.

The factors (F-*x*) are explained and sometimes organized in categories (C-*x*) to provide a better understanding of their relationships and of their context. They have also been categorized and sorted as inhibitors, enablers, inverse factors and ambiguous factors, according to the kinds of influence presented in the paper (Fig 1.); categories are shown in the right margin using category abbreviations on the form YYY-Z with meanings given by Figure 1 below. For further explanations of the meaning of these categories, please refer to the paper.

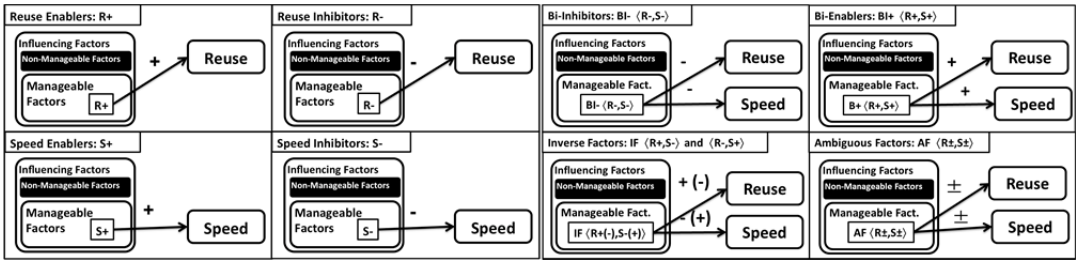


Figure 1 Kinds of Influence

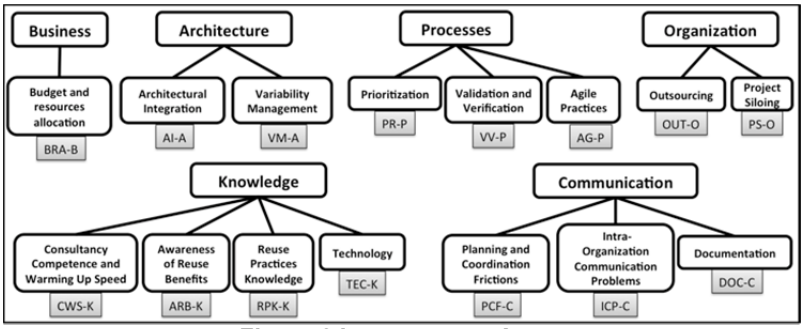


Figure 2 Improvement Areas

Reuse Inhibitors (Case A)

C-1: Coordination Business side – Development side.

F-1. Communication barriers business side – development side ICP-C

F-2. Different views between business side – development side BRA-B

F-3. Business side afraid of upfront investment for an SPL BRA-B

F-4. Business side not aware of SPL benefits ARB-K

F-5. Customers of different locations don't understand the benefits of a SPL ARB-K

F-6. Rush to hit the market (focus only on first deployment speed) BRA-B

These 6 factors are very connected among them, so we have grouped them in the cluster *C-1: Coordination Business side – Development side*. From many points in the interview is clear that the interviewee (the architect) is willing to implement an SPL but he is worried about the agreement on the investment from the business side (F-3). This is also connected with the fact that the previous attempt to implement it was not successful and the business side was more focused on the single delivery instead of on the development of common software among the products. Obviously, this also means that business side and development side have different views (F-2) on software development key factors, the former one focusing on 1st deployment speed (F-6) while the other takes more into account the replication speed and the benefits of having a common platform (which leads to F-4). The interviews suggest also the existence of barriers in communication, since it seems that development side receive orders from different masters (from different projects (F-7)) rather than being part on the decisional process on what development strategy would be better to manage the whole line of products. These factors have obviously a negative effect on considering reuse practices as important, which means that they hinder the development of a reuse culture and reuse practices (so they are Reuse Inhibitors).

F-7. Lack of awareness of the architecture benefits in the business side BRA-B ARB-K

This is another factor explicitly mentioned by the architect. To achieve a 1st deployment speed the management is keen to sacrifice (in terms of budget and time-resources) the development of a carefully thought architecture. This hinders reuse practice, since a well-defined architecture would create a suitable environment for reuse.

F-8. Lack of SPL knowledge in some parts of the organization (including some developers) ARB-K

While the architect claims his knowledge and willing of implement a SPL, this is hindered by the lack of basic concepts, principles and technical skills throughout the whole organization. This results in decisions (management side) and implementations (development side) not in accordance with the SPL principles (and reuse in general).

C-2: Early branching hinders SPL implementation

F-9. Different products initially managed with a branch, then it became too late to merge the branches into a SPL **RPK-K**

F-10. Different products of a SPL ended up to be entirely distinct products

The architect described how creating an early branch in configuration management contributed to keep the products separated hindering their synchronized development and therefore the implementation of reuse strategies.

F-11. Not sufficient early architectural planning **PR-P**

This is connected with the cluster *C-1* explained before. Not planning for architecture implies not planning for reuse. However, "not sufficient early architectural planning" has different causes that don't necessary involves the lack of resources and understanding from the business side (as explained in *C-1*), but could be the consequences, as mentioned in the interview, of inadequate practices in prioritization and lack of knowledge in architecture design. Moreover, this is a problem also connected with the implementation of Agile Methods, which in general are keen to pay less attention to develop architecture in advance, but focusing on accurate refactor afterwards. Unfortunately, it seems that usually only the first part of the principle is implemented (early architectural planning), while refactor post-iteration is usually not applied due to lack of time, resources and prioritization goals.

F-12. Developers and Architects have an intuitive way of evaluating costs **ARB-K**

Asking for cost evaluation, Developers and Architects have an intuitive way of evaluating costs and no helpful decision tools. This reduce their awareness of the benefits that software reuse could bring.

C-3: External Market Constraints against SPL

F-13. Market constraints too strong against the products synchronization of the SPL ***Context**

F-14. Different time constraints between different markets ***Context**

The interviewees stressed their problems in adapting the intended SPL to markets that are not synchronized. Some of them could require a certain TTM, while others don't. However, an important market demanding and early release could force the focus on the 1st deployment speed (F-7), strongly hindering the implementation of the SPL (since, as pointed out in F-11, once a product is branched from the rest it's very likely that it would remain independent from the SPL)

Reuse Inhibitors (Case B)

C-4: Projects are too independent

F-15. Few and big projects are not synchronized

F-16. Project-related bugs (even in the same reused component) PS-O

The projects are quite isolated, causing a phenomenon called "siloing": every decision aims for the local optimum (F-113), and this obviously hinders the development of common assets among the projects.

F-17. Business side lacks awareness of SPL benefits ARB-K

See factor F-5.

C-5: Communication Problems Business side – Development side

F-18. Communication barriers business – developers. Evaluation of costs and feasibility of new features depends on implementation ICP-C

F-19. Communication barriers business – developers. Wrong assumptions on what can be reused and adjusted for an upcoming product (causing inaccurate budget allocation) ICP-C

These two factors are focused on the communication barriers existing between the business side and the development side (see also cluster *C-1* for case A). Here the granularity of the factors is finer. The business side made the customer promises based on wrong assumptions (regarding implementation issues and reusability of existing software). This is caused by a lack of consulting of the development side, which cannot give feedback about the feasibility of the features and about the reusability of the components. Having wrong estimation hinders the ability of planning for reuse.

C-6: Reuse implementation

F-20. Reuse not supported from the PL - individual initiative BRA-B ARB-K

F-21. No time for product development (project transversal) PS-O

F-22. Reuse is not organized RPK-K

F-23. Too much application of opportunistic reuse RPK-K

In one of the analyzed sub-case the Product Line Management (PLM) is not supporting reuse with budget allocation or the introduction and support of good practices (F-23, F-26). This is a key factor for enhancing reuse, since PLM would be the right position for evaluating needs across products of the same line. This reuse Inhibitor is supported also in another interview, where a PLM has managed to keep the focus on reuse allowing the development of substantial portion of software (F-78). However, it has to be said that the latter developed product contained less variability along time, making reuse much easier. In the former case, reuse was limited to individual initiative: the effect was that a "light framework" (citing the interviewee) of libraries was reused occasionally and opportunistically (F-27), which results to be not very effective in the end. On a wider scope, the interviewee mentioned a

general lack of time (not included in the resource allocation by the management, F-23) to work on software shared among projects.

F-24. Budget dedicated only for a specific business goal (0% to architecture) **BRA-B**

This factor is connected with both F-6 and F-7 of case A. The architecture is not taken in consideration as a business goal (and neither is reuse). This implies the lack of awareness of benefits of architecture and reuse on the business side.

F-25. Hard to predict where the variation points will be in many years **VM-A**

It's difficult to plan reuse when the variation points are unknown. This factor becomes especially impacting when many years pass between the developments of two projects involving the same product (context related problem). The risk is to waste a lot of time on implementing variation points that will be useless in the next product.

F-26. Loss of knowledge about the reused light framework variation points created many years before **VM-A**

The practice of reusing a light framework of libraries in an opportunistic way (see C-6 explanation) led to lose the knowledge of the implemented variation points.

F-27. Developers too constrained by system engineers on design **ICP-C**

The interviewee stressed this point very much. If reuse is not taken in consideration by System Engineers and their specifications are too specific on a design level, implementing reuse becomes very hard because of the mismatch between the different designed systems. At the same time, another interviewee (a PLM that managed to reuse a substantial part of the software across different products, see C-6 and F-23 explanation) stressed the same point as important to implement successful reuse.

F-28. Different favorite languages or tools in the same team **TEC-K**

Different favorite languages or different tool chains inhibit reuse. The lack of compatibility among languages or tools, in fact, makes it harder to adapt an artifact to a different environment.

F-29. Consultancy lacks of competences **CWS-K**

Connected with F-9. The interviewee highlights the lack of competence as a reuse inhibitor. In the specific case, a small component has to be completely rewritten because of the poor implementation. The interviewee then specifies that this is usually a problem derived from working with consultants, who often don't have the competence for a specific task.

RPK-K

F-30. Non-flexible data structures hinder reuse

The interviewee explains a situation in which [language name] data-structures are not flexible and therefore not easily extensible (every time a new field is added everything has to be recompiled).

F-31. Not explicative documentation hinders understanding the code

DOC-C

The interviewee stressed the importance of the "quality" of the documentation, i.e. well written to *help understanding* the code, rather than on its volume and on the details on *describing* the code.

Reuse Inhibitors (Case C)

C-7: Products Synchronization to adopt a shared component

F-32. Different platforms in different products AI-A

F-33. Internal interface exposure (slight documentation provided) VM-A

F-34. One of the products was very old and stable: it had to replace the working part with the new common component AI-A

F-35. One of the products had a longstanding, dated process of verification and validation which suffered when applied to the new component VV-P

A shared component has been adopted by three existing products. The products had different platforms, and especially for one of them this created problems. The variability was not limited to features but spread also to the platform adaptation (F-35), which decreased the common part resulting in extra-work, extra-communication and extra-issues. One of them was the exposure of an internal interface (F-36). For one of the products, in order to address some issues during the product adaptation (F-37), had to access an internal interface, supposed to be used only inside the teams. The documentation for such interface was scarce, therefore a lot of extra communication took place. The mentioned product, being old and stable, had a longstanding, dated process of verification and validation, which suffered when applied to the new component: for example, they had many test cases to be carried out manually. Since the common component had to be integrated, this meant that a lot of test cases had to be changed and adapted and new ones had to be written and carried out manually.

F-36. After the forced introduction of the common component, projects were afraid to integrate new common components because of the unpleasant experience ICP-C

This is the interviewee's perception (common component development unit) about the reasons why they have communication problems with the receiving products. After the problems caused by other descripted factors (i.e. *C-7*), other projects were disincentivized to accept the same approach to reuse. Moreover, the adoption of the shared component was a decision coming from "higher plans" and appeared more like a forced obligation rather than an agreed solution.

Speed Inhibitors (Case A)

F-37. Long warming up periods for consultancy **CWS-K**

Complexity of the system (F-62), tool, and lack of detailed documentation (see F-115) cause a long (1-2 years) warming up period for consultancy.

F-38. Lack of support for changing interfaces **ICP-C**

The interviewee mentions that they have decision support, i.e. for bugs fixing, but they don't have decision support for changing interfaces, which is crucial for SPL.

F-39. Test team delay caused by lack of early integration tests **VV-P**

Early integration tests are not in place enough (too waterfall-like), slowing down the test process.

F-40. Testers located in separate team **VV-P**

The interviewee mentions many disadvantages of having testers located in a separate team. Speed is inhibited, since the early integration tests are less implemented, causing a lot of re-work to integrate the individual parts of the system.

Speed Inhibitors (Case B)

F-41. Standard project structure don't fit some kinds of projects PS-O

The standard structure for projects is designed for the substantial ones, but cause overkill for small projects.

F-42. Customized agile shift from waterfall process, but still not completely in place AG-P

The shift from the waterfall process to Agile methods is partial, since only some practices have been implemented. The interviewee perceives this as an issue for reaching the right speed.

F-43. Focus on product scope instead of on time of deployment - long iterations PR-P

A negative practice for speed is to assign a considerable workload to developers without prioritizing, causing in fact long iterations. The conversation suggests that this is usually done to assure that they have their work and do not have to be managed for a (relative) long time. This practice postpones checkpoints, causing the early implementation of non-important features which impact 1st deployment speed.

Speed Inhibitors (Case C)

F-44. Decisions from systemization take long time ICP-C

The development unit responsible for the shared component had to wait a long time before receiving the specification from system engineers. As for the causes of this, the interviewee suggests the presence of a considerable amount of variability to satisfy many customers (not necessarily connected with the developed component).

C-8: Projects synchronization on the common component

F-45. Synchronization problems between the projects to adopt the common component PCF-C

F-46. Lack of will from some projects to adopt the common component PCF-C

F-47. Communication barriers of the common component development unit with the integrating projects ICP-C

F-48. Very long decision process for the common component to be accepted in all projects PCF-C

The development of the component required a lot of communication and synchronization among the different projects. Different requests and availability caused many delays (F-50, F-53). The interviewee expresses her perception of a lack of will from some projects to adopt the common component (F-51), which created barriers in communication (F-52). Probable causes for these were the fact that some of the products already

had in place another working component with similar functionalities (F-71, F-112) and that the projects were forced to integrate the component (F-72, F-39).

F-49. Text documents as artifacts from the systemization ICP-C

The interviewee mentions the use of not appropriate documents for specifications.

F-50. Lack of use cases VV-P

The architect of the unit developing the shared component recognizes the issue of not utilizing use cases.

C-9: Integration of the Common Component

F-51. Validation of integration of the functional independent components consumes a lot of time AI-A VV-P

F-52. Time-consuming match of the pre-existent abstract model with the real equipment VV-P

F-53. Layer architecture against functional domain of the component caused delays in verification and validation of the component against all the layers AI-A VV-P

F-54. Difficult integration of the common component into different architectures of different projects AI-A

Validation and integration of the component was particularly slow. In fact, the component was developed as a functional one in order to start a process to switch from a layer-oriented architecture to a functional-oriented one (F-73). The component had to be adapted to the existing architecture, and part of the validation time had to be spent on matching the pre-existing abstract model with the real equipment of the component.

Bi-Inhibitors (Case A)

F-55. Big complex model, hard for new employees and new consultants to understand if the functionalities are already implemented CWS-K

The dimensions of the product (and of the model) hinder the recognition of a specific functionality: if a new employer or a consultant want to reuse it, they first have to look for it, which takes a lot of time.

C-10: Distributed teams coordination

F-56. Problems with the attitudes and values of distributed teams PCF-C OUT-O

F-57. Satellite unit is "invisible" and not directly controlled PCF-C OUT-O

F-58. Satellite unit auto prioritizes, have different masters (different goals) OUT-O

When covering the outsource topic, the interviewee mentioned that when a satellite team has different attitudes and values with respect to the teams working in house, this usually created a lot of problems in synchronizing priorities, coordinating work and agreeing on interfaces. In fact, the satellite unit followed different masters and therefore auto-prioritized its work. This is a problem when align the roadmaps and to coordinate for a reuse strategy that involves satellite units and to drive the unit to implement the most important features.

F-59. Consultancy has knowledge of others' ways of working but don't have specific knowledge of the product. CWS-K

One issue mentioned by the architect was that sometimes consultants bring new and interesting ideas coming from their background in other organizations, but they lack the specific knowledge of the product. This hinders reuse because the domain knowledge is important to understand well what can be designed to be reused and what is not. Besides, the lack of specific knowledge causes a long warming-up period and sometimes poor implementation choices (F-29).

Bi-Inhibitors (Case B)

F-60. Developers and system engineers not co-located – problems in requirements agreement **ICP-C**

System engineers were not co-located creating many problems. Requirements are delivered instead of communicated and discussed, and clarifications regarding some of them are usually more difficult if the responsible is not available. Speed is inhibited, since it can happen that some non-important requirements are fulfilled causing a lot of extra-work. Reuse is inhibited, too, since system engineers' design should include software reusability as a criterion, which is a knowledge coming from the software development side.

F-61. Lack of incentives on reuse – hour-based wage doesn't incentive reuse **BRA-B**

The interviewee finds that hour based wage doesn't incentive reuse. In fact, reuse practices decrease the working hours of employees, who don't have any reason to avoid it.

F-62. Lack of prioritized backlog **PR-P**

A prioritized backlog is necessary to focus on the right features for the 1st deployment speed. Moreover, it is important to incentivize architecture work in order implement reuse.

F-63. Processes were chosen to fit project management, developers would prefer a different way of working **AG-P**

The interviewee's perception is that the processes were chosen without trying to follow developers' ways of working, hindering their speed (for example, forcing too much administrative documenting) and not promoting reuse.

Bi-Inhibitors (Case C)

C-11: Common component Variability Management

F-64. Variability management – agreement problems between projects and common component on the delivered feature set PCF-C VM-A

F-65. Variability management – projects had to verify unused features VM-A

F-66. Variability management – lack of feature management

F-67. Lack of hardware variability management VM-A

F-68. Lack of variability modeling VM-A

The shared component was designed with the same features for every product. No variability modeling was used (F-58). The internal customers (products) didn't agree on the desired common set of features (F-56), for example because they didn't want to test functionalities that they didn't use (F-57), the amount of communication and frictions was increased.

F-69. A common component between the different products was forced, so other projects didn't really want to use it. ICP-C

See cluster C-8 and F-38. This factor has been categorized as Bi-Inhibitor because, given the other mentioned factors, forcing the integration of the common component produced both speed and reuse issues.

F-70. "White box" specification from systemization ICP-C

The interviewee mentioned how System Engineers usually delivered over-detailed specification documents ("white box"): this way, architects and designers were usually constrained, struggling to plan reusable artifacts.

Reuse Enablers (Case A)

F-71. Reuse of models **RPK-K**

Both interviewees state that the reuse is carried out with models and clearly imply that modeling is a good reuse practice.

F-72. Different backlogs for functionalities and architecture **PR-P**

Architecture work is prioritized and not left aside. This is important to support reuse.

F-73. Use models to handle variability **VM-A**

Models are also used to handle variability. This helps organizing the common parts and recognizing reusable software.

Reuse Enablers (Case B)

F-74. Frequent meetings and agreement between Developers and System Engineers discussing the benefits of Reuse (different unit)

ICP-C
ARB-K

- The interviewee, having different roles such as developer, architect and product line management, revealed that part of the system has been successfully reused. He stressed the importance of having a good and continuous discussion between developers and system engineers. When asked, he also said that the practice of continuous communication needed much time (more than 10 years) to become effective. Moreover, it's clear from this interview that the Product Line Manager has supported reuse. This is on the same line of what has been said another interviewee from case B (see cluster C-6, F-30).

Reuse Enablers (Case C)

F-75. Long term plan: having more common components integrated in the different products **ARB-K**

The interviewee revealed that the common component has been the first of a planned process to switch from a layer-oriented architecture to a functional components-oriented one (more reuse friendly). Many issues related to case C can be viewed under this light: somewhere the process had to be started to support future reuse, and probably this was a first step to promote the switch, which means that the problems may disappear or decrease when a different environment has set up. This may suggest that the introduction of reuse may require a long process to be effectively supported.

C-12: Successful platform variability handling

F-76. Variability - different adaptors for different platforms of the projects **VM-A**

F-77. Product B and product C using the same platform **AI-A**

Three products adopted the common component. Two of them (that we called B and C) used the same one (F-80), helping the integration of the shared software, while A used a different one, which caused problems (see also cluster C-7). However, the platform side variability in the common component was well managed, with the introduction of platform-adaptors (even though there were drawbacks, see C-7).

F-78. Successfully integration of the common component **AI-A**

The eventual integration of the common component showed that the strategy was successful, even though some approaches caused (especially) speed problems (see C-7 and C-11). However, this doesn't necessary mean that the those approaches cannot be changed to improve speed without hindering effective reuse.

F-79. Variability management - build time scripts to include the code **VM-A**

Even though variability management lacked in general (see F-40, C-9), build-time scripts were used to include the code.

Speed Enablers (Case A)

F-80. Co-located teams **AG-P**

For both the interviewees the co-location is an important factor for speed, especially to avoid frictions, synchronization delays and communication problems.

F-81. Weekly prioritization of the functionalities with the customer **PR-P**

The Agile principle of continuously consulting the customer has been implemented, which helps the teams to focus on the key features.

F-82. Short period problem dealing **AG-P**

The interviewees mention some Agile approaches whose implementation is still in progress. Those practices should help to quickly deal with problems. Some examples are daily meetings, daily deliveries to testers and the goal of having spread releases during the iteration (8-10 weeks).

Speed Enablers (Case B)

F-83. Changing programming language improved productivity TEC-K

The interviewee mentioned the decision to switch from Ada to Java, which increased the productivity of the developers.

Speed Enablers (Case C)

C-12: Agile environment supported implementation speed

F-84. Cross-functional teams AG-P
F-85. Informal roles, knowledge sharing AG-P
F-86. Teams have the same responsibilities, same backlogs PR-P
F-87. Very short time of implementation TEK-K

The interviewee praised the implementation of some Agile methods (F-89, 90, 91). In fact, the implementation time was very short (F-93).

F-88. Common component architecture free from other projects AI-A

The unit developing the common component had the freedom to use its preferred architecture. This was a good choice for the fast implementation, even though created integration problems. However, it has to be viewed in the light of F-79.

Bi-Enablers (Case A)

F-89. Incentive from management side to developers' proactivity: communication of small improvement through meetings ICP-C

The interviewed product line manager revealed that the management side is encouraging developers to communicate every small improvement that can be implemented. This is connected also with

F-90. Model driven code generation TEC-K RPK-K

The interviewees were quite convinced that code generation was a good practice both for speed and reuse.

Bi-Enablers (Case B)

F-91. Open Source products easier to manage than outsourced component **OUT-O**

The interviewee was very pleased by his experience with open source components. Support was always very fast and effective, and obviously it saved time from developing the same component in house. He mentioned these benefits also in comparison with outsourcing.

F-92. Reuse of the design for small components **RPK-K**

The interviewee mentioned a small component that he wanted to reuse. A consultant had developed it in a dated language, and the result was poor. However, the design was clear and easy to understand (also because of the limited size of the component), so the interviewee decided to recode it in a newer language, but following the design. In the end, the coding part resulted to be very quick. This can be considered as a good reuse practice: reusing the design for small components could result in having the advantages of reuse at a relative low cost in terms of speed.

Bi-Enablers (Case C)

F-93. Developers have the will to take part in the systemization ICP-C

The developer with the architect role revealed that the development unit would like to participate in the systemization to give their contribute and at the same time to give suggestions from their point of view.

F-94. The common component uses only a small part of the platform functionalities - not affected by platform changes AI-A

The common component doesn't rely heavily on the platform functionalities, having a small amount of dependencies from it. Changes on the platform don't affect the component much, and it doesn't require to be adjusted, saving extra-work.

F-95. Successful integration in product C - no previous similar component to be replaced AI-A

The integration of the common component in product C did not cause problems. This suggests that if the same strategy would be chosen in future, the approach could be successful.

Reuse Enablers & Speed Inhibitors (Case A)

F-96. Functionality requests from different projects PCF-C

Receiving functionalities specification from different projects helps planning for reuse. However, if the projects are not well coordinated and synchronized (depending also on the markets), this practice brings a high penalty in terms of speed. On the contrary, focusing in the 1st deployment speed for one project could obstruct the application of reuse practices.

Reuse Enablers & Speed Inhibitors (Case B)

F-97. Conservative mindset, tend to keep old assets – this often needs extra code to handle it PCF-C

The interviewee mentions a certain conservative mindset present in the company that tends to keep old assets. This in theory would help reusing, but in fact it often means extra code to handle old code.

F-98. Massive use of documentation DOC-C

Large amount of documentation helps reuse, however it takes time to be written, and it's usually difficult to produce it in the right way much time after the development.

Speed Enablers & Reuse Inhibitors (Case A)

C-13: Prioritization determines the focus of the team

F-99. Architectural improvements are done by the most available and / or most skilled developers PR-P

F-100. Customer functionalities preferred by developers over architecture PR-P

F-101. Relocation of developers from other tasks to follow customer prioritization PR-P

The management of human resources has an impact on reuse and speed. In case A we can observe a high focus on speed (especially the 1st deployment speed): F-104 shows that architectural improvements are left to the most skilled developers, when available, otherwise only to the available ones (also suggested by F-106). Furthermore, developers like more to work on customer functionalities rather than on architecture (F-105).

F-102. Short lifecycle of the product *Context

The lifecycle of the product is around 7 years. Thus, the focus is pointed on the fast delivery (1st development speed) rather than on the maintenance-related advantages given by reuse.

Speed Enablers & Reuse Inhibitors (Case B)

F-103. Risk in planning reusable components - lost time for the first client and having the wrong variation points in the future VM-A

The interviewee identifies this way of thinking in his environment. If they plan a reusable component very well they have to detract time from the first delivery to gain in future assets. Moreover, variation points could be wrong and the asset could result to be non-reusable.

Reuse Enablers & Inhibitors (Case B)

C-14: Reuse practice: a light framework of libraries

F-104. Reuse of a light framework of utilities extracted from the previous project RPK-K

F-105. Problems splitting reused framework RPK-K

F-106. The reuse of a light framework leads to have a lot of copy paste RPK-K

A light framework (for example a set of libraries) has been extracted from a product to be reused and expanded in a successive one (F-109). However, after some applications of this schema, the framework grows wild, which means that it has to be split and re-organized. This risk to nullify the advantages gained so far (F-110).

Reuse Enablers & Inhibitors (Case C)

F-107. First common component shared among several products – component already present in some of the products **ARB-K**

See explanation in C-8, F-79 and C-11.

Speed Enablers & Inhibitors (Case B)

F-108. Problems with team synchronization - low inter-team communication (very independent teams) **ICP-C**

Teams are very independent. This decreases the amount of communication (and the problems derived, such as delays, synchronization), but at the same time they pay a "fee" at integration level. It's hard to say which one is impacting speed more, even though the interviewee seems keen to prefer early communication.

Other Ambiguous Factors (Case A)

F-109. Some components outsourced **OUT-O**

The experience of the interviewees with outsourced component is mixed. Outsourcing avoids in-house development, which is good for speed. However, communication is often a problem, slowing down the process anyway. As mentioned in F-12, C-10 and F-96, other problems derive from outsourcing. The advantages in term of reuse are dependent on the kind of component outsourced.

F-110. Lack of detailed documentation **DOC-C**

See also F42, F-103, F-41 and F-34. Detailed documentation takes time to be written, could improve reuse but only if well explaining the code, and could also decrease the warm-up process for new employees and consultants.

F-111. Improvements depend on leaders mindset (open to listen and recognize strengths and weaknesses) **ICP-C**

This is an "in vivo" quotation from an interviewee from the case A. He was stressing the attention on the importance of a leader when new or improved practices have to be implemented. This is especially important when there are small teams and it would be desirable to introduce cost- and speed-effective practices that require coordination (like reuse).

F-112. The same people implement functionalities and architectural improvements **AG-P**

The pros and cons in terms of speed and reuse are many and blurred. In case A, the interviewees said that the same people are developing customer-oriented functionalities and architectural improvements: in fact, there is not a clear distribution of such responsibilities. As mentioned in C-13, this has mainly a good effect on speed and not on reuse. However, having both knowledge (features and architecture) and a careful prioritization mechanism (internal or external) could really help to implement reusable software.

Other Ambiguous Factors (Case B)

F-113. Projects focused on the local optimum

PS-O

Many factors in case B are connected with this trend cited by the interviewees (F-20-24). The right evaluation of costs and the budgets allocation are obviously very important to reach business goals, and the local optimum is the first step. Interviews show that a broader view is necessary to understand the importance of reuse, which could increase the local expenses and the 1st deployment speed of the current projects, but would be important to decrease costs and speed afterwards (of the same project and of the others). At the same time, speed is very important, and 1st deployment speed could be decisive also to keep a good relationship with the customers.

F-114. Documentation slows code improvement

DOC-C

The interviewee mentioned this factor as improvement inhibitor. The idea is that, once a substantial amount of documentation has been written for some software, it's difficult to change the code to improve it if the developer has to rewrite a lot of documentation. The categorization is motivated by both negative and positive influence in reuse: in fact, less improvement means less chances that a piece of software is adjusted for reusability. At the same time, as specified before, a well-documented functionality enhances its future reuse. As for speed, documentation slows down the 1st deployment speed (and included improvements), but has a positive impact on the time spent on warming up for new employers and consultants (reducing the 1st deployment speed) as well as on the time necessary to find and reuse a functionality in opportunistic reuse.

Reuse with Speed – Interview Protocol

INTERVIEW GUIDELINES
PRIMARY QUESTIONS
FOLLOWUP QUESTIONS

- What is Being Reused
- Effects of Reuse
- Reuse => Reduced Speed
- The Reuse Ecosystem
- Modes of Reuse
- Organizational Boundaries Influence on Reuse / Speed
- Engineering Communication Infrastructures Influence on Reuse / Speed

INTERVIEW GUIDELINES

These interview guidelines are based on template provided by "Washington School Research Center"

The protocols that follow include open-ended interview questions and a number of subject areas to keep in mind. If these areas are not addressed by the open-ended responses, some of the suggested probes might be helpful.
The purpose of probes in interviews is to enable the person being interviewed to be as informative as possible in their responses. They are neutral prompts that encourage additional information, but do not suggest specific answers. Some examples of probes are "How is that?" or "In what ways?" and so on.
The protocols below include some follow-up questions that might be helpful for obtaining further information when probes do not result in covering the areas. Since follow-up questions should touch on whatever the interviewee has already said, there is no best way of phrasing them. These are only suggestions.

INTERVIEW CHECKLIST

• Start each interview with a statement ensuring confidentiality. That interview will be recorded, what will happen to the recording (transcription by you), who will be able to see the recording (only you and the research team), and how it is going to be stored (on password protected USB disks not connected to the University network).

• Have the informant sign an agreement form. (See sep. document.)

• Interview one project in each organization (Volvo, Ericsson, Saab)

• For each project, interview one *line-manager* (or equivalent role) and one *team-leader (*or equivalent role).

• Plan for 1h with the informant, but allow yourself at least one more hour (to make room or continued discussion.)

• Plan an additional hour *directly after the interview* to write down your thoughts, and to add meta-data to a *case-study database*

• Always test the technical equipment. (Recording device, before the interview and during the interview. Test your laptop as backup.)

• This is basically an inductive approach. Don't be too focused in questions—ask general, open-ended questions.

• Start interview with general questions about *role* and *experience* from agile methods and experiences from reuse. (This makes the informant more comfortable, and is important in analyzing answers later.)

• Use 2 or 3 open-ended questions to get the interview started.

• Bring along an easily navigable presentation with suggestive pictures and questions.

• Develop three or four questions that 'get at' each of the phenomena.

• If you already covered a question in your discussion, *don't ask it again*; rathert summarize what the informant already said, and ask if there's something else to add on that topic.

• Use same questions at each site: major questions, followed by minor ones

• PROBE (i.e. *what things have you been doing in the past that allow you to...;* use follow up questions (i.e. '*You haven't mentioned...',)*

• Bring along a .ppt with thought-provoking pictures, such as current WoW, envisioned WoW. Metaforic pictures (such as the one from Gulliver's travels) are very effective in probing

• Be ready to show pictures with key phenomena studied *if the informants asks; precisely* what do *you* mean by reuse / speed /other phenomena. (This presentation grows gradually.)

PRIMARY QUESTIONS

1. Tell me about your role.

2. Q: What experiences do you have from reuse?

Probes:
a) What processes did you use? (Ask an example?) [**PROCESSES**]
b) How formal were these process? [**FORMAL**]
c) How strictly were they followed? [**STRICTLY**]

d) How was variability managed? [**VARIABILITY**]
e) How did you decide what to reuse? (see also 8) [**OPPORTUNITIES, DECISION TOOL**]

f) How did you manage backward compatibility? [**BACKWARD COMP.**]

g) What tools did you use to manage reuse? [**TOOLS**]
h) Did you have some tool to show you economical consequences of your chosen reuse technique? [**DECISION, ECONOMIC, TOOL**]

i) How successful were the projects? [**SUCCESS**]
j) What step will be next? [**NEXT STEP**]
k) What is the "lesson learned"? [**LESSON LEARNED**]

3. Q: What experiences do you have from agile methods?

Probes:
a) At which scale were methods used? [**SCALE**]
b) How long were the sprints? [**SPRINTS** LENGHT]

c) How did a team development roadmap influence other team's roadmap (time consuming)? [**ALIGNEMENT ROADMAP**]

d) How was LSV managed? [**LSV**, LATEST SYSTEM VERSION]

e) How did the team prioritize their work? Did you have any problems with that? [**WORK PRIOR., INFLEX./INERTIA**]

f) Did teams act somehow in an unpredictable way? [**UNPREDICTABLE ADOPTION**]

g) What was the role of architecture? [**ARCHITECTURE**]
h) How was architecture documentation represented? [**ARCH**. **DOC**.]

i) How was intra-team communication organized? [**INTRA**-TEAM COM.]
j) Inter-team communication? [**INTER**-TEAM COM.]

k) Let's say that a team needed a change that involved other teams: how did they act? [**ENFORCE CHANGES**]

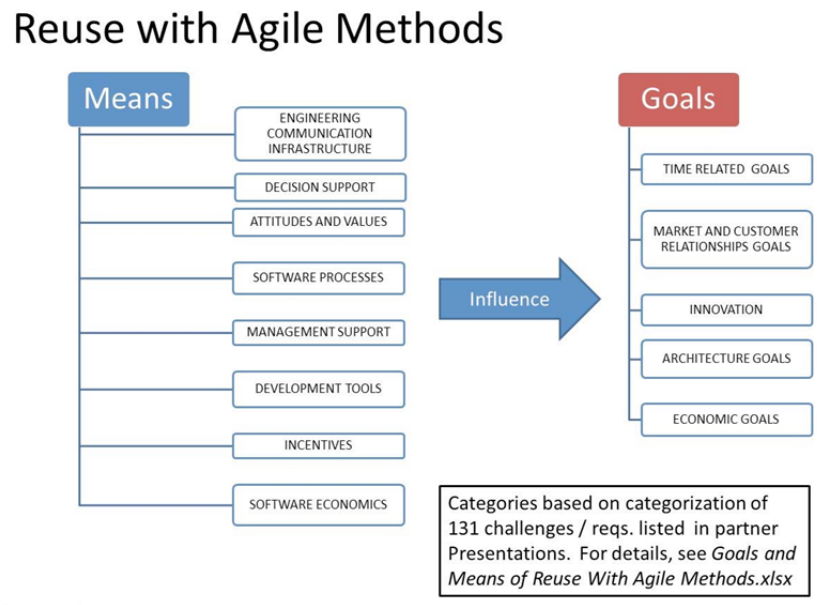
4. *What challenges do you see in having reuse* within the context of <whatever method is in use at site>?

5. *What challenges do you see in increasing speed* in the context of <whatever method is in use at site>?

6. Here is a list of improvement areas in which we have recognized challenges in combining reuse and speed.

7. *Q: What challenges in combining reuse with speed do you see in these areas?*

8. Q: What challenges, in these areas, do you think that others may see?



9. Q: Can you think of further improvement areas (important to increased reuse and speed) that are overlooked in this picture?

10. <u>Final Question Q:</u> Is there anything else you would like to add?

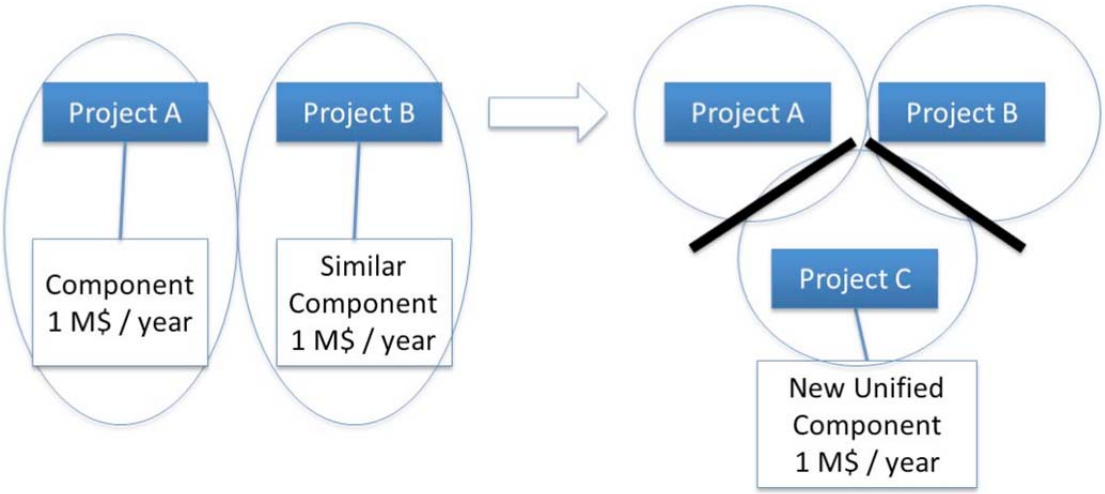
FOLLOWUP QUESTIONS

Specific areas for follow-up questions
(N.B.: These are leading question, but good probing for further details. Only to be used to dig deeper, not to provoke a course change.)

What is Being Reused?
You mentioned that X (= Components, Subsystems, etc.) is reused. Are you aware of any other things reuse, e.g.,
-

Effects of Reuse

This picture shows an idealized form of reuse.
Can you comment on how this falls out in practice? (The Black Bars are organizational boundaries.***)***

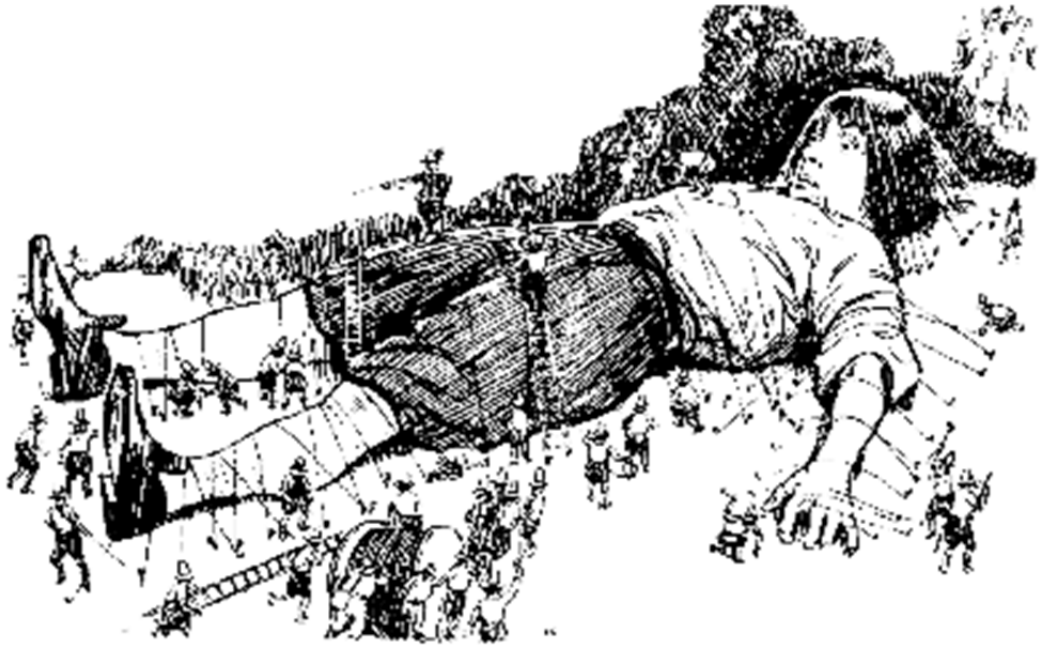


□

Reuse leading to Reduced Speed

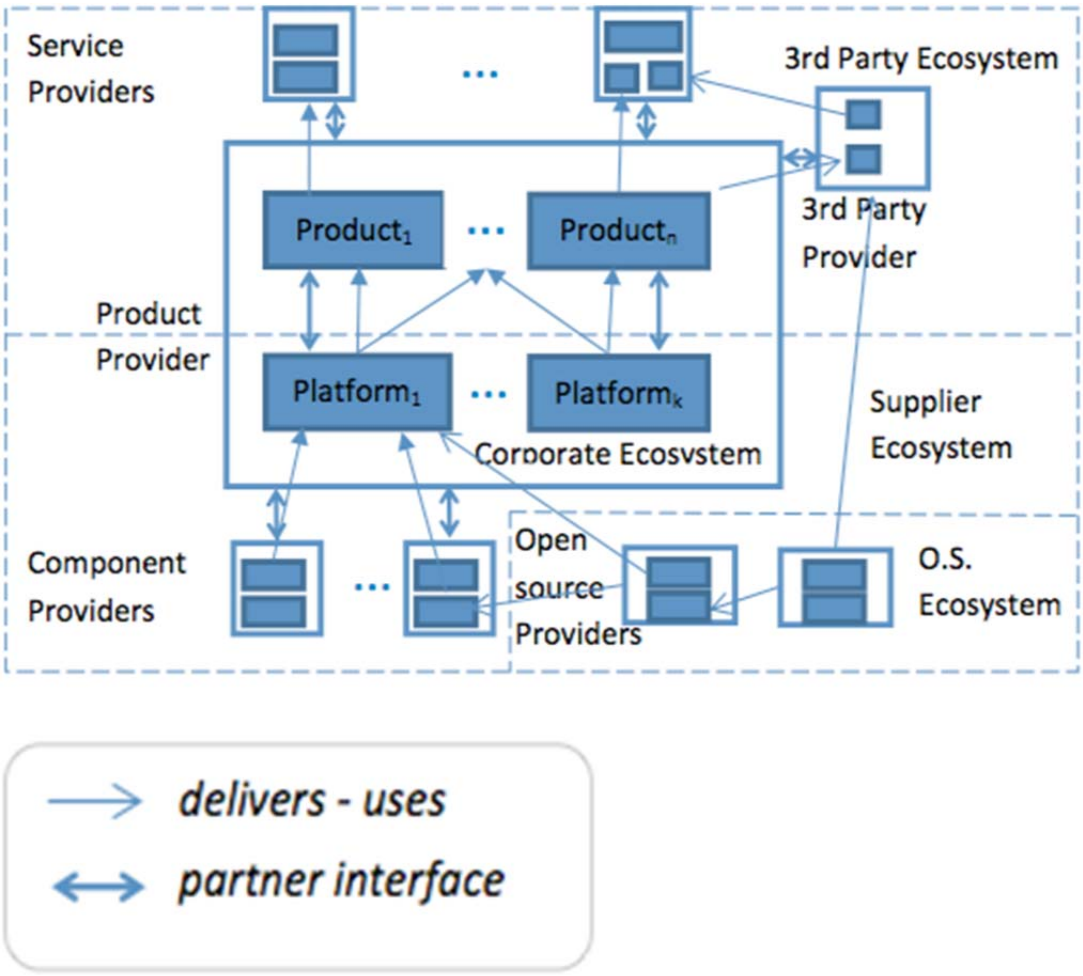
SPLE => Reduced Agility => Reduced Speed for New Product Offer

You mentioned that SPLE/Reuse/XXX may hinder agility. Can you further elaborate on that? Could you elaborate on that on the basis of this picture?



Reuse Ecosystem

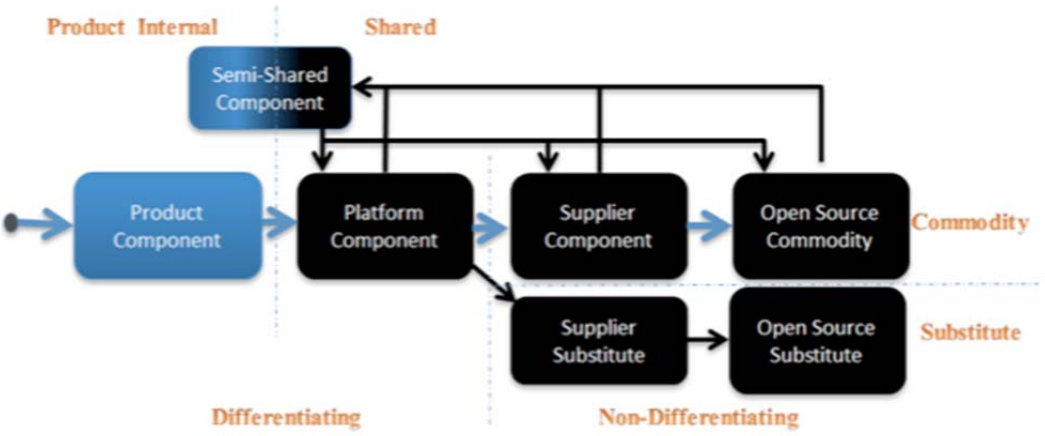
You mentioned various forms of reuse (i.e., X,Y), were used. How do they relate to the following picture?



Reuse Modes of reuse for software components

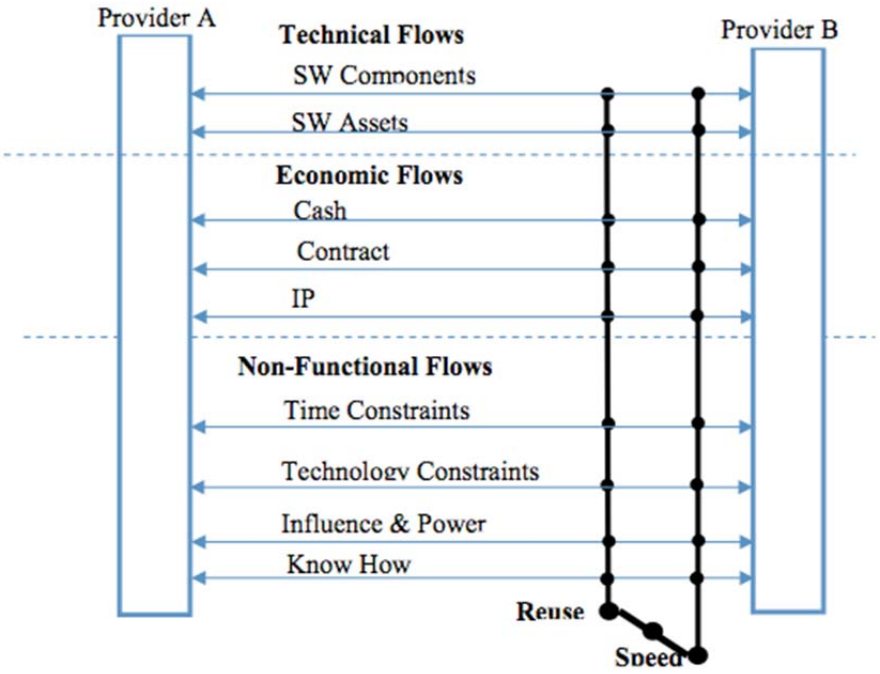
You mentioned that reuse changed throughout a lifecycle. How do changes relate to the following picture?

Can you recognize some forms of reuse not present in this picture?



Organizational Boundaries Influence on Speed

You mentioned that reuse is hindered by X. How does that relate to this picture?



Seeing this picture, are there any other things that you can think of that red*uce speed?*

*Inter- and Intra-Organizational Boundaries Influenc*e on Speed

You mentioned that speed is hindered by X. How does that relate to this picture? How does that relate to this picture? <Picture Above>

Seeing this picture, are there any other things that you can think of that inhibit or restrict reuse?

Communication Infrastructures Influence on Reuse

You mentioned communication between X and Y to be a major inhibitor for speed / reuse? Could you elaborate on that on the basis of this picture.