Coordination in Large-Scale Software Teams

Andrew Begel, Nachiappan Nagappan
Microsoft Research
Redmond, WA, USA
andrew.begel@microsoft.com, nachin@microsoft.com

Christopher Poile
Edwards School of Business
University of Saskatchewan
Saskatoon, SK, Canada
poile@edwards.usask.ca

Lucas Layman
National Research Council
Ottawa, ON, Canada
lucas.layman@nrc.gc.ca

Abstract

Large-scale software development requires coordination within and between very large engineering teams which may be located in different buildings, on different company campuses, and in different time zones. From a survey answered by 775 Microsoft software engineers, we learned how work was coordinated within and between teams and how engineers felt about their success at these tasks. The respondents revealed that the most common objects of coordination are schedules and features, not code or interfaces, and that more communication and personal contact worked better to make interactions between teams go more smoothly.

1. Introduction

Coordination between software development teams is one of the most difficult-to-improve aspects of software engineering. Kraut and Streeter argue that the software industry has been in crisis mode for its entire existence, and a root cause is the difficulty in coordinating work between teams of developers [9]. Researchers have studied professional software development teams empirically to gain greater understanding of how software development processes, tools, and people impact coordination. The importance of intra- and inter-team coordination is a foremost concern as software development increasingly becomes globally distributed, and remains a persistent challenge in other disciplines as well.

To understand inter- and intra-team dependencies in large-scale software development, we conducted a web-based survey of 775 Microsoft developers, testers and program managers. We asked engineers how they coordinate tasks with teams they depend on and with teams that depend on them, and how they communicate with their dependencies when things go wrong. We then asked how developers feel about working with dependent teams to understand where they would like to see improvement. We described some aspects of this research in prior work [1].

We find that it is important to consider the different roles that people play on their teams when coordinating with others. Processes and tools intended for software developers may not be appropriate for program managers. The two job roles "live" in different applications in their daily work; tools intended for one role's applications just may not be used by the other. In addition, intra-team and inter-team coordination communication modes are very different. It may be easy to pay a personal visit to a team member who likely sits on the same floor as you, but much more difficult and socially awkward to visit a collaborator from another team, especially when that collaborator is not a friend. We also find that the overhead of communication and maintaining relationships between individuals who coordinate on different teams is high, but necessary to getting work done successfully. Many respondents to our survey wished they could get more information and action about their dependencies with less active communication requirements. Even though coordination is difficult, we find that even in a cross-section of one of largest software companies in the world, almost all engineers are required to coordinate with others to get their work done.

2. Survey Design

The research was conducted using an anonymous web-based survey offered over a period of two weeks in Au-

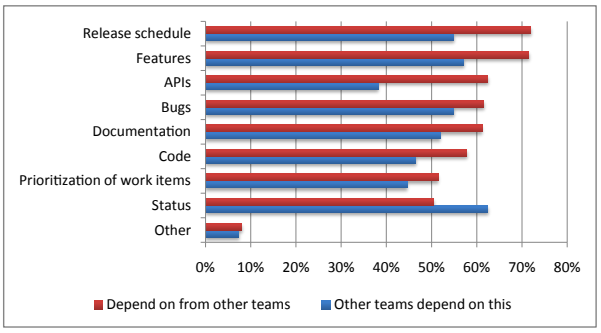

Figure 1. What engineers depend on from other teams, and what other teams depend on from them.

gust 2007 inside the Microsoft Corporation. An invitation was sent by email to 2,535 developers, testers, program managers (PMs), architects and user experience engineers, consisting of a 10% random sample of employees in each job role. At Microsoft, program managers gather customer requirements, write design specifications, and manage the project schedule. Developers turn design specifications into implementation specifications, and write the code. Testers turn design specifications into testing specifications, write test harnesses, and conduct black box testing. Architects do long-range and product-wide planning, and user experience engineers design user interfaces and conduct usability studies. Respondents were offered a chance to win a single $250 gift certificate as incentive for completing the survey.

The survey questions were divided into three sections: demographics, details about how coordination occurs in one's team, and perceptions of how well coordination was practiced within one's team and its dependencies. A Likert scale of "All of the time, Frequently, Occasionally, Rarely, Almost Never, N/A" was used. All of the survey questions reported on in this paper can be found in Appendix A.

3. Data and Results

We received 820 responses, of which 45 were invalid (we removed duplicate and empty surveys), for an overall response rate of 30.6% (775 / 2,535). In our sample of 775 respondents, 39.2% are developers, 33.6% are testers, 20.3% are program managers, and the rest (6.9%) have other job roles. 76.5% of respondents were individual contributors; the rest (23.5%) were leads or managers. Respondents had an average of 9.6 years (SD: 6.3) of experience as software engineers, and spent 5.0 years (SD: 4.1) working at Microsoft.

We asked developers how they depended on other teams and how they coped when dependencies went awry. The first question asked survey recipients what artifacts they depended on from other teams, and what artifacts other teams depended on from them. The responses are shown in Figure 1. The respondents report 72% depend on another team's release schedule and 71% depend on the features of another team's product. At a slightly lesser level are APIs (62%), bugs (62%), documentation (61%) and code (58%). Prioritization of work items and status are around 50%. When considering what other teams depend on from them, however, status becomes the most frequent, rising to 62%, leaving features (57%), bugs (55%), release schedules (55%) and documentation (52%) less frequent. Less than half of the respondents say that other teams depend on their code (46%), prioritization of their work items (45%) or their APIs (38%).

When asked how they kept track of the work items they depended on from other teams (shown in Figure 2) most respondents (69% overall) reported using email. 61% use a work item database and 56% talk about them at status meetings. After that there is a large drop to keeping track of dependencies in your head (38%) followed by using Outlook tasks (30%), Sharepoint websites (29%), using a point person in charge of keeping track of dependencies (27%) and Excel spreadsheets (26%). Keeping a list on paper, text editor or personal whiteboard follows. Note that only 16% of people keep track of work items outside their team by reading the source code checkin messages, and very few (2%) keep track of their work items on a public whiteboard.

Notice, too, that program managers (PMs) use a greater diversity of tools to keep track of dependencies, almost 10% greater in many tools than developers and testers. The biggest disparities are that PMs use Sharepoint websites twice as much as developers, and use Excel spreadsheets three times as much as developers. Developers use source code checkin messages 60% more often than PMs or testers.

Also illuminating are the ways to how engineers communicate to unblock themselves from a critical dependency (shown in Figure 4). When the dependency is *within* their team, almost all (89% and 88%) respondents said they would send an email and pay a personal visit to the person blocking their work. Instant messages and phone calls came in a distant second place (55% and 54%), followed by a setting up a one-time meeting (45%), escalation to their manager (38%), and communicating via work item database (38%). When the dependency is *outside* their team, almost all still use email (89%) to communicate, but paying a personal visit drops to 48%. Instead, people use the phone (59%), escalation to a manager (52%) or one-time meetings (51%) as a substitute.

From the data in the chart, it appears that respondents would use a point person to keep track of dependencies more for external rather than internal collaborations. In addition, having a manager talk to another group's manager is

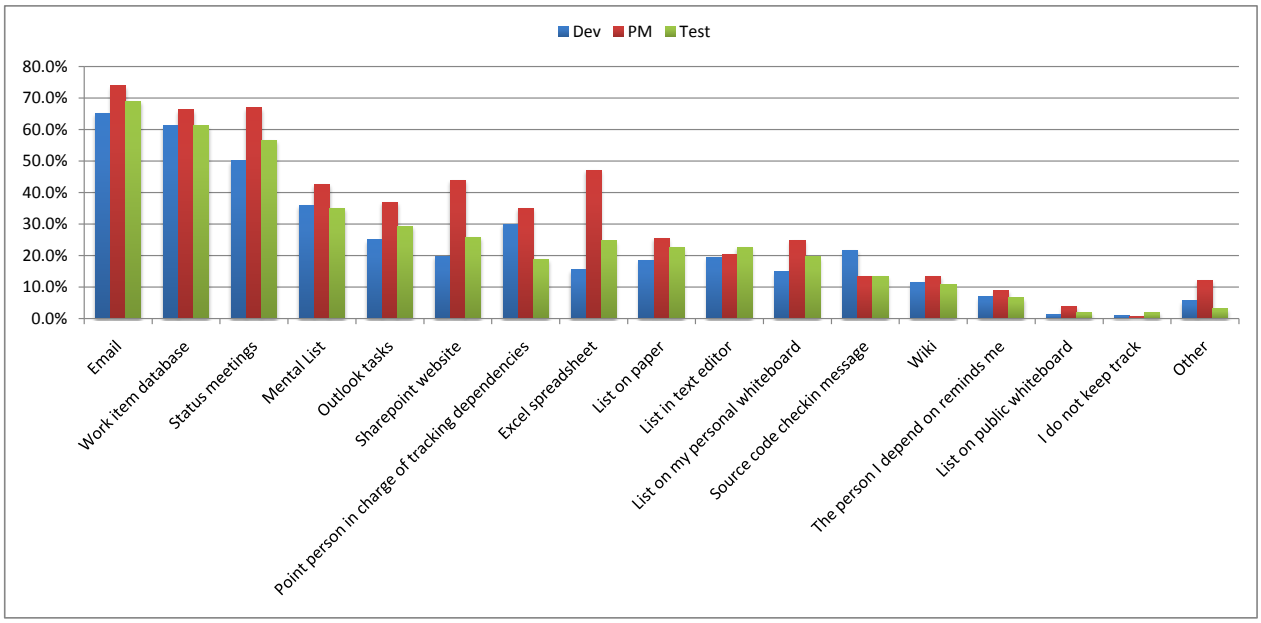

Figure 2. The tools that engineers use to keep track of dependencies on other teams, divided by job role. The 'other' category is not shown for clarity of presentation.

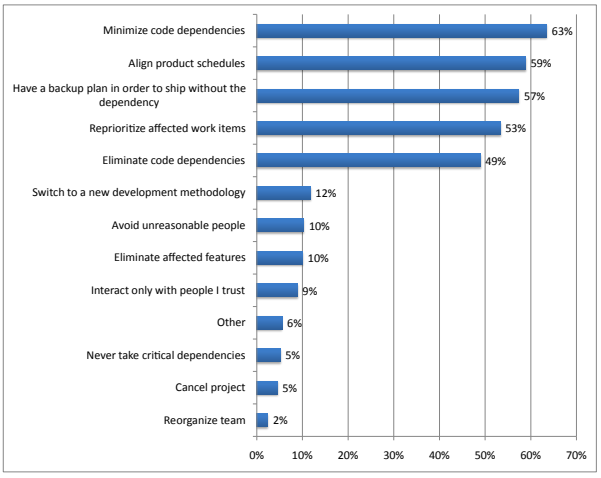

Figure 3. How engineers mitigate (anticipated) problems with dependencies on other teams.

much more common with external dependencies than internal ones (25% vs. 8%).

Notice also, that 98% of respondents depend on people outside their teams (only 2% report that they do not have any dependencies in Figure 4).

Since not all collaborations with other people and teams go as planned, we asked participants how they mitigate anticipated and/or real problems with their dependencies. There were five strong responses (shown in Figure 3, three that minimize the dependency itself, and two that adjust the project schedule. 63% of respondents would minimize all code dependencies on other teams. 59% would align their product's schedule with their dependencies' schedules in order to ship only when their dependencies have finished their own work. This can be problematic if a dependency slips their schedule. 57% of respondents say that their team makes sure to have a backup plan to ship their own product without the problem dependency. 53% would reprioritize their work items, potentially slipping a feature or work item to the next release. 49% would eliminate all code dependencies, which could mean to "clone and own" (copy and paste) the desired functionality from the other team's codebase. All of the other responses, including canceling the project (5%) and reorganizing the team (2%) are much less frequent.

Communication overhead and maintaining relationships takes a big toll on coordination practices and effectiveness (see Figure 5). Almost 50% of respondents say that they need to proactively ask their colleagues for status frequently or all the time. Lack of communication causes problems as well for 23% of respondents who need frequently or all the time find that work items they depend on have changed without any notification. When communication does occur, 25% of respondents say it is frequently or always difficult to get a team they collaborate with to implement a change

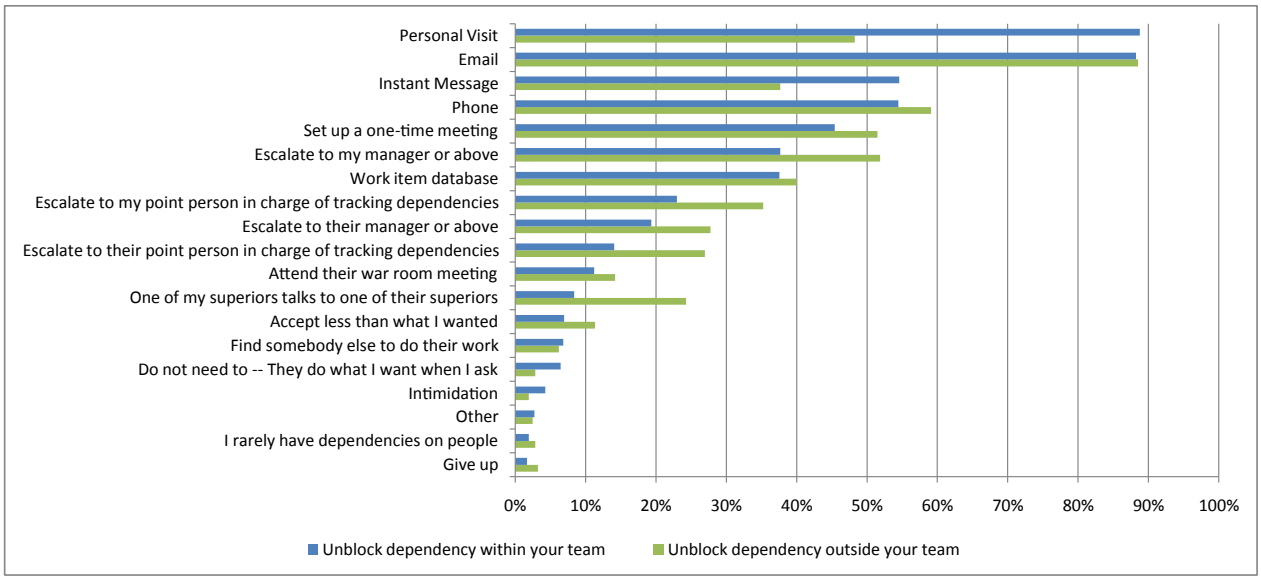

Figure 4. How engineers communicate with one another when they need to unblock themselves from a critical dependency.

they require. 48% of respondents find that must maintain constant contact with the team they depend on is the way to get what they want. This contributes greatly to overhead in depending on other teams. It is also probable that much of this overhead is due to other team's deprioritizing their dependent's work items. Only 30% of respondents say that their dependencies frequently or always tell them where their needs fit into their dependencies' priorities. This lack of information often causes anxiety in teams that have many of these dependencies. Even worse, if a team had a choice of teams to depend on, they would certainly choose the ones that place them number one on their priority list, rather than a team that served too many masters.

A solution that many teams have found to work best for them is to maintain personal connections with the people who work in the teams they depend on. 87.6% of respondents agree with the statement "I feel that having personal connections with teams that I depend on is helpful to me."

4. Discussion

From the survey results in Figure 1, we can see that most respondents work on teams that consume software from others. This is consistent with the demographics of Microsoft's software teams. As a company, Microsoft creates platforms and applications that depend on those platforms. There are necessarily fewer platforms than applications. What was surprising from the survey responses is that most teams depend on other teams' release dates and features more than other teams' code or APIs. This implies that teams worry more about a feature shipping on a particular date than the details of how the functionality will be implemented.

The teams that provide libraries offer status as their main consumable, followed by features and schedules. Without being privy to the inner workings of other teams' processes, status updates are one of the only ways for a team to manage its dependency on another team's software prior to the ship date. Will they finish in time? Will the requested features and work items be completed? The lack of this information can make team leads anxious and unable to mitigate the problems associated with teams that may not deliver what was promised. We see these problems in the perception data (see Figure 5) where pinging people for status occurs quite often. Respondents also noted that work items they depended on were changed without any change notification. We presume these were not changes to say that the work items were done early or with more features than requested. The change in status is one thing, but the lack of notification of the change aggravates whatever anxiety the dependent teams were already feeling about their dependency.

Another surprising result from the data (shown in Figure 2) is that the fourth most popular way to keep track of dependencies is to use one's memory. In cases where engineers have few and infrequent dependencies, this may work just fine. We do not have data to support or refute this. We hope that when engineers must maintain more significant relationships between their team and others, they use more robust forms of written material. The third most popular tool is a status meeting, usually occurring face-to-face

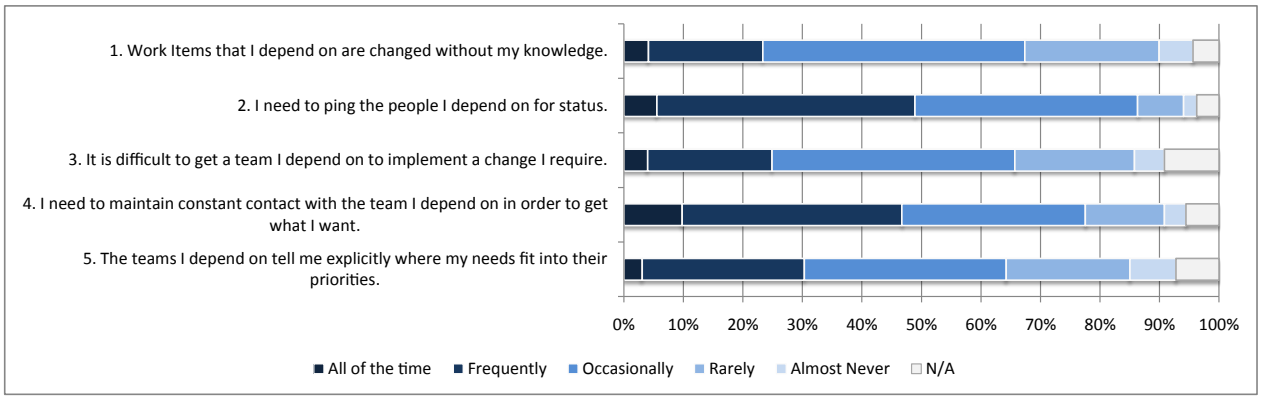

Figure 5. How engineers perceive the success of coordination communication.

or held via audio or video conferencing. This corroborates with the perception data (Figure 5), where respondents report that in order to get what they want from another team, they must maintain constant contact with them. Status, bug triage, and similar kinds of meetings are held frequently by teams to manage their workflow. Attending another team's meeting can give one great insight into what their priorities are and how they evolve over time.

We present the data in Figure 2 split by job role to illustrate the different distribution of answers. Program managers (PMs) at Microsoft handle requirements, scheduling and coordinate feature and work item completion with other program managers, developers, and testers. Thus, it should be unsurprising that they use tools to track dependencies simply more than developers or testers. However, PMs' greater dependence on tools is important for tool builders who aim to help resolve coordination problems between teams. These tools will likely be most effective if they "live" in the same applications that program managers already find themselves in. From the survey data, we can see that this is often not in the code, itself. Thus, to most effectively resolve team coordination issues with a tool, it is important for tool builders to note that non-programmer PMs ought to be their target audience, rather than the developers who create the code or the testers who validate it. We can make this argument stronger by noting again that the main objects mediated in a dependency are features and product schedules, both items maintained by program managers, not by developers or testers.

From Figure 4, we can see that the way coordination occurs within teams is very different than between teams. Within teams, personal visits and email are the most popular ways to fix blocked dependencies. The parties know one another and are comfortable visiting in person to explain their issues and get them resolved. When dependencies cross teams, however, the frequency of in-person visits drops by almost half. It is likely quite awkward to pay a visit to someone whom one does not know, and interrupt their work (which hopefully is unblocking the issue at the same time, but probably not) for something that may not be immediately important to the person. But from the other survey responses, we can see that attending another team's status meeting is fairly accepted practice, and thus may be a good substitute for a personal visit. As can be observed in Figure 4, almost all of the answers were chosen more frequently by respondents with external dependencies than those with internal dependencies. Thus, the diversity of methods to unblock oneself is high, along with the associated communication overhead required to avail oneself of them all. This communication overhead likely contributes to feelings that more work is required to manage external dependencies than internal ones.

Finally, over half of the respondents report that their teams have contigency plans to deal with unfulfilled dependencies, either delaying, replacing, or canceling requested functionality. Only 5% report that their projects are canceled for the lack of the dependency, thus, the projects reported about by these respondents appear modular enough to adapt to unanticipated failures of collaboration between teams. If, as reported in the perception data in Figure 5, more teams would tell their dependents explicitly where they stood in the team's priorities, perhaps the numbers of projects that are delivered with less functionality than desired could be reduced by teams seeking out other less burdened teams to meet their needs.

5. Threats to Validity

Our survey was conducted at Microsoft Corporation; while we imagine its results apply broadly to software developers at other companies, studies at other sites would be useful to highlight behaviors which may be affected by Microsoft norms and culture.

6. Related Work

Coordination problems are inherent in any sort of distributed work situation. Many researchers have looked into coordination problems in software development organizations and discovered difficulties caused by geographic distance [7, 10], team size [5], lack of personal contact [8], lack of awareness [6], poor knowledge flow and communication breakdowns [2], and architectural modularity [3]. Kraut and Streeter [9] found that developers preferred to communicate frequently and informally to coordinate with one another on schedules, bugs, tests and design reviews. de Souza and Redmiles [4] surveyed developers and catalogued how they managed dependencies, both for potential problems with coordination that they expected to experience and problems they expected to cause for others.

7. Conclusion

From these survey results, we can learn some lessons about how to improve intra- and inter-team coordination. While most teams experience challenges in coordination, the majority appear to get their tasks done, albeit with less completed than they had hoped for. We saw that examining the needs and methods of engineers with different job roles can help to focus process changers and tool builders on less obvious audiences for their interventions, hopefully to effective results. Another takeaway message from our paper is that creating and maintaining personal relationships between individuals on teams that coordinate is indicated by many respondents as a good way to successfully collaborate with colleagues. However, while more communication between teams can help improve coordination, it can also increase process overhead. Creating tools that engender the right kinds of communication, such as automatic status gathering and change notification could help people keep up with their dependencies more efficiently.

For as long as specialization and collaboration has been around, many possible solutions to coordination problems have been tried and will continued to be invented. The data in this paper helps identify some of the potential target areas and audiences for such interventionary solutions.

References

[1] A. Begel. Effecting change: Coordination in large-scale software development. In *Workshop on Cooperative and Human Aspects of Software Engineering*, pages 17–20, Leipzig, Germany, May 2008. ACM.

[2] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[3] C. R. B. de Souza, D. Redmiles, and P. Dourish. "breaking the code", moving between private and public work in collaborative software development. In *Proceedings of GROUP*, pages 105–114, Sanibel Island, FL, 2003. ACM Press.

[4] C. R. B. de Souza and D. F. Redmiles. An empirical study of software developers' management of dependencies and changes. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 241–250, New York, NY, USA, 2008. ACM.

[5] J. Fred P. Brooks. The mythical man-month. In *Proceedings of the international conference on Reliable software*, page 193, New York, NY, 1975. ACM Press.

[6] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *Proceedings of CSCW*, pages 72–81, Chicago, IL, 2004. ACM Press.

[7] J. D. Herbsleb and R. E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of ICSE*, pages 85–95. IEEE Computer Society Press, 1999.

[8] P. Hinds and C. McGrath. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In *Proceedings of CSCW*, pages 343–352, Banff, Alberta, Canada, 2006. ACM Press.

[9] R. E. Kraut and L. A. Streeter. Coordination in software development. *Communications of the ACM*, 38(3):69–81, 1995.

[10] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering Methodology*, 11(3):309–346, 2002.

Appendix A: Survey Questions

1. Which of the following items do you depend upon from other teams? i.e. you need the following from them in order to complete your task, or a change in the following with affect your task.

   (a) APIs
   (b) Code.
   (c) Release schedule
   (d) Bugs
   (e) Features
   (f) Prioritization of work items
   (g) Status
   (h) Documentation
   (i) Other

2. Which of the following items do other teams depend on you to provide? i.e., they need the following from you in order to complete their task, or a change in the following will affect their task.

   (a) APIs
   (b) Code.
   (c) Release schedule
   (d) Bugs
   (e) Features
   (f) Prioritization of work items
   (g) Status

(h) Documentation
(i) Other

3. How do you keep track of the items that you depend upon outside your team?

(a) Mental list
(b) List on paper
(c) List in text editor
(d) List on my personal whiteboard
(e) List on a public whiteboard
(f) Excel spreadsheet
(g) Source code checkin messages
(h) Outlook tasks
(i) Work item database
(j) Email
(k) Sharepoint website
(l) Wiki
(m) Status meetings
(n) Point person in charge of tracking dependencies
(o) The person I depend on reminds me
(p) I do not keep track
(q) Other

4. When you get blocked on a critical dependency *within* your team, in what ways do you communicate to unblock yourself?

(a) Personal visit
(b) Email
(c) Phone
(d) Instant Message
(e) Work item database
(f) Set up a one-time meeting
(g) Attend their bug triage meetings
(h) Escalate to my manager or above
(i) Escalate to their manager or above
(j) Escalate to my point person in charge of tracking dependencies
(k) Escalate to their point person in charge of tracking dependencies
(l) One of my superiors talks to one of their superiors
(m) Intimidation (i.e. pay a visit and do not leave until you get what you want)
(n) Find someone else to do the work
(o) Accept less than what I wanted
(p) Give up
(q) I rarely have dependencies on people within my team
(r) Do not need to – They always do what I want when I ask
(s) Other

5. When you get blocked on a critical dependency *outside* your team, in what ways do you communicate to unblock yourself?

(a) Personal visit
(b) Email
(c) Phone
(d) Instant Message
(e) Work item database
(f) Set up a one-time meeting
(g) Attend their bug triage meetings
(h) Escalate to my manager or above
(i) Escalate to their manager or above
(j) Escalate to my point person in charge of tracking dependencies
(k) Escalate to their point person in charge of tracking dependencies
(l) One of my superiors talks to one of their superiors
(m) Intimidation (i.e. pay a visit and do not leave until you get what you want)
(n) Find someone else to do the work
(o) Accept less than what I wanted
(p) Give up
(q) I rarely have dependencies on people within my team
(r) Do not need to – They always do what I want when I ask
(s) Other

6. What do you do in your own project to mitigate the effects (potential or actual) of depending on other teams?

(a) Minimize code dependencies
(b) Eliminate code dependencies (e.g. “clone and own”)
(c) Never take critical dependencies
(d) Align product schedules
(e) Reprioritize affected work items
(f) Avoid unreasonable people
(g) Interact only with people I trust
(h) Cultivate in-person relationships (e.g. become friendly, put a face to the name)
(i) Switch to a new development methodology (e.g. Scrum, Feature Crews)
(j) Have a backup plan in order to ship without the dependency
(k) Cancel the project
(l) Reorganize the team
(m) Other

7. Likert-style questions. Answers: All of the time, Frequently, Occasionally, Rarely, Almost Never, N/A.

(a) Work items that I depend on are changed without my knowledge.
(b) I need to ping the people I depend on for status.
(c) It is difficult to get a team I depend on to implement a change I require.
(d) I need to maintain constant contact with the team I depend on in order to get what I want.
(e) The teams I depend on tell me explicitly where my needs fit into their priorities.