AMAZON
SOFTWARE DEVELOPMENT ENGINEER (SDE)
ROLE GUIDELINE

Guideline Last Updated: June 13, 2019
Contains expectations for: Individual Contributors only. Managers use the SDM Role Guideline.
Guideline Notes: For questions about the SDE Role Guideline, contact sde-level-clarification@amazon.com. This guideline contains **Alt-Text** for images to support screen readers.

Amazon Software Development Engineer (SDE) Role

This guideline contains general expectations for the SDE role. It describes the most common responsibilities, however given the wide variety of businesses and technologies at Amazon, it cannot capture all expectations. No two teams are alike and each is encouraged to develop their own approach to delighting their customers. This alters the way an SDE is expected to operate and what constitutes success.

Each section has a specific purpose:

- **Section 1: "SDE Level Matrix"** is a high-level view of how the functional dimensions map to each SDE level. It is for quick comparisons of level vs. role expectations in hiring debriefs, performance, and promotion discussions.
- **Sections 2-5:** "**What you do**" contains the most detail about SDE job level expectations to guide team hiring strategies and performance discussions. Contains a graphic to illustrate the scope and influence for the level.
- **Sections 2-5 "Moving to…"** criteria isolate key skills at the next level that should be demonstrated to be considered for promotion. This is not a promotion checklist. Every promotion case is unique; the results you deliver (and *how* they are delivered) also play a role in promotion evaluations.

This guideline does not repeat expectations documented in previous levels (i.e., the abilities of higher levels inherit those of lower levels). This means a Principal SDE has all of the abilities described in Levels 4-7.

1. SDE Level Matrix

The purpose of this matrix is to provide a quick view of how SDE functional dimensions change by level. It does not include **Amazon Leadership Principles**, as they do not change by level. For more detailed guidance, see **sections 2-5**.

1.1 SDE Functional Dimensions

- **Ambiguity**: The degree that software **project**[1] and/or **product**[2] requirements are defined. Also clarifies how much supervision is required vs. when SDEs can deliver independently.
- **Scope and Influence**: The type of work typically handled by each SDE level (**components**[3], **features**[4], **architecture**[5], etc.). Clarifies SDE levels that influence within a **team**[6] versus (vs.) an **organization**[7].
- **Advises:** Who SDEs typically collaborate with and influence.
- **Execution**: Concise view of expectations. Effectiveness of software delivery – as evidenced by timeliness, quality, usability, stability, security, performance. As levels get higher, the mix of work shifts from **tactical**[8] toward **strategic**[9].
- **Technical Complexity**: Types of technical problems/efforts that each level can reasonably handle. Includes types of trade-offs an SDE at a particular level may make.
- **Impact**: Customer experience. Software size and importance. Business Impact.
- **Process Improvement**: On development (e.g., packages, build, testability, release/deploy), customer experience (e.g., accessibility, localization, self-service), operational excellence (e.g., software resilience, maintainability).
- **Experience**: Recommendation to meet level expectations. A college degree or formal training is **not required** if the candidate has equivalent knowledge gained from experience.

---

[1] **Project** - a point-in-time endeavour with an end date, undertaken to create a unique product, service, or result.
[2] **Product** is any good or service that Amazon sells to customers or offers internally for use by the company or its employees.
[3] A software **component** is a building block that does not solve a customer or business problem on its own, but is part of a solution. A piece of technology with a clearly defined interface that is indivisible, has minimal internal dependencies, and deliverable (somewhat) independently.
[4] A **feature** is any functionality or cohesive set of functions that has value to a customer.
[5] **Architecture** is a set of components and features that delight customers or solve a problem. Design includes when to build, refactor, or deprecate.
[6] **Team** is defined as the group reporting to a line manager (e.g., classic two-pizza team with < 10 people)
[7] **Organization** (Org) - a group of teams that together provide one or more product(s) or business function(s).
[8] **Tactical** is when someone takes action(s) to achieve *current* goals and prioritize based on a limited time horizon. The employee is capable of making short-term trade-offs (e.g., time vs. quality), but may not understand the long-term effects of their decisions.
[9] **Strategic** is when someone takes action(s) to achieve *future* goals. The employee can define the problem or opportunity and align others to their vision/solution. Implementation requires some level of judgement and expertise to make appropriate trade-offs.

1.2 SDE Level Matrix

Dimension	L4: SDE I	L5: SDE II	L6: SDE III	L7: Principal SDE
Ambiguity	Works on defined technology projects. Will occasionally need guidance.	Technology strategy is defined. Solution design is not. Delivers independently, but will seek some direction.	Business problem is defined. Technology strategy may not yet be defined. Delivers independently, with limited guidance.	Business problem and architectural strategy may not yet be defined. May not know what the problem is before starting. Drives clarity. Delivers with complete independence.
Scope and Influence	Work is typically focused on software components. May work on small features and develop small tools.	Work is typically focused on software features and major portions of team software. Influences within own team.	Work is typically focused on team architecture/ product solutions. Influence may extend to related teams, but only in the course of a project.	Work is typically focused on organization architecture/ product solutions. May focus more narrowly based on skill/expertise. Influences business and technology direction.
Advises	Peers	Peers, Manager, Program or Product Manager (PMs)	Peers, Manager, PMs, Sr. Manager	Director (may VP) , Broader Tech Community
Execution	Develops, tests, and deploys software. Work is tactical. Learning design approaches. Contributes to team planning and design discussions. Delivers high-quality, correct production code. Can troubleshoot software failures. May be part of an oncall rotation. Escalates when projects hit roadblocks and risks. May train interns.	Designs, develops, tests, and deploys software. Work is tactical. Understands a broad range of design approaches. Clarifies requirements. Assists with coding/story estimates. Able to review code and provide constructive direction. Identifies and resolves root causes. Mitigates immediate risks. Decides if they can handle or need to escalate. Begins to mentor.	Designs, develops, tests, and deploys software. Can lead large projects. Knows how to divide so they can work in parallel with other SDEs and reassemble into a cohesive launch. Work is tactical and strategic. Proactively simplifies code and resolves team architecture deficiencies. Learning to **force multiply**[10]. Mitigates long-term risks. Finds a path forward in difficult situations. Actively mentors. Performs SDE tech assessments.	Designs, develops, tests, and deploys software. Can lead strategically important projects involving multiple teams. Effective force multiplier. Can deconstruct an architecture to be developed by different teams. Work is strategic. Aligns teams toward simple, coherent designs. Handles escalations and mitigates complexity risks (deconstructs into appropriate simplicity). Actively develops and mentors others. Performs PE promo assessments.
Technical Complexity[11]	Solves **straightforward**[12] software problems. Solutions are tested (continuously testable, when possible) and may need refinement.	Solves **difficult**[13] software problems. Solutions are logical, testable, maintainable, and efficient. Makes technical and design approach trade-off decisions (application level)	Solves **complex**[14] software problems. Solutions are extensible and scale. Removes bottlenecks. Makes trade-offs: short-term vs. long-term technical decisions.	Solves **significantly complex**[15] or endemic software problems. Solutions are exemplary in terms of robustness, stability, scalability, cost-effectiveness. Makes trade-offs: opportunity vs. architectural complexity.
Impact	Impacts software quality and customer experience (CX) of product features.	Impacts software quality, CX, and maintainability of a product/features.	Impacts software architecture, dependencies, performance, and business value.	Impacts how their organization operates. Impacts software offerings and architectures of the teams they influence.
Process Improvement	Improves testability, operational excellence (OE) metrics, and team documentation.	Drives best practices. Improves the speed of delivery. Automates tests and manual OE tasks.	Leads projects to streamline team processes. Creates mechanisms to reduce churn on implementation.	Sets the standard for engineering, excellence. Drives best practices across organization.
Experience	Meets SDE I Tech bar	3+ yrs. SDE experience	5+ yrs. SDE experience	10+ yrs. SDE experience

[10] **Force multipliers** are individuals who either possess attributes or deliver solutions that dramatically increase (hence, multiplies) the effectiveness of a group, giving them the ability to accomplish greater things.

[11] **Hiring Managers and Tech BRs:** see the SDE Tech Bar level in **FAQ #4** at the end of this guideline

[12] **Straightforward** problems/efforts have minimal visible risks or roadblocks. *What* to accomplish is clear (and does not appear to be complicated), but *how* to accomplish that work is not clear.

[13] **Difficult** problems/efforts have visible risks or roadblocks; requiring skill and a considerable amount of work to resolve/deliver results. SDE examples may include refactoring portions of software, large migrations, etc.

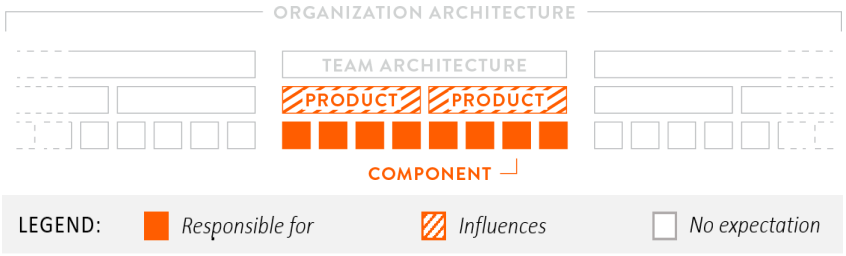
[14] **Complex** problems/efforts have visible risks, roadblocks, and **constraints.** SDE examples: legacy dependencies, data quality, compliance (e.g., financial legal, regulatory agencies), architecture or technology limitations, extensibility asks (e.g., enable clients to customize behavior without direct work from an engineering team), performance (e.g., implementing an API with requirement of p99.9 < 200ms is a lot harder than p99.9 < 2 seconds), and scalability (e.g., designing a payment processor connector that processes 2M messages per hour is a lot harder than 2K messages per hour).

[15] **Significantly Complex** problems/efforts have substantial risks/constraints and they **conflict** with each other. Not all constraints or risks are apparent. Requires enough expertise to 'see around corners' and to navigate incredibly complicated situations.

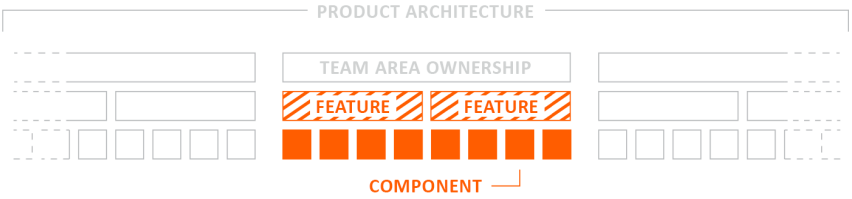
2. SDE I

2.1 Scope and Influence

SDE scope depends on the organization architecture, products, and whether they are owned entirely by one team or built by several teams. To visualize scope, there are two image examples. The first applies to a Two-Pizza team that owns software that provides automation for a business offering. These teams often own multiple tools and services end-to-end.



The second example applies to teams that own a single piece of technology or a set of features that are part of a large product architecture that is spread across multiple teams. In these cases, the SDE's team does not own the full product architecture. They own portions of it.



In either case, SDE I scope of work is at the small to mid-size component or small feature level. Their influence is within their own team.

2.2 What you do…

You use technology to solve straightforward problems, seeking input and guidance from team members. You may create or have responsibility to improve or invent small tools or applications. You are able to take a defined design and turn it into code and deliver it on schedule, applying appropriate technologies and current software engineering best practices. You write secure, stable, testable, maintainable code with minimal defects. You are proficient in a broad range of data structures and algorithms, knowing when it is appropriate to use them (and when it is not). You make appropriate implementation trade-off decisions (e.g., array or hash table). You do not put the company at risk (e.g. pulling in unlicensed code, working on code in unsafe ways, etc.). You participate in team design, scoping and prioritization discussions. You seek to learn the business context and technologies behind your team's software. You work effectively with customers and/or internal partners to understand business impacts and identify any opportunities/problems arising from technical decisions. You invent, refine and develop your solutions to ensure they are meeting customer needs and team goals. You are a passionate advocate for your customer.

You assume responsibility for the state of the code you both inherit and produce. You get your designs and code reviewed. You test your code thoroughly. You classify, store, and handle data in accordance with Amazon policies. You track security risks and mitigate and/or escalate them in a timely manner. You understand the maintenance characteristics, runtime properties, and dependencies of your team's software, including hardware platform, operating system and build dependencies. You clearly document your software to ensure that future generations of developers understand the intention behind the features and components you build. You may be part of an oncall rotation. In the event of a problem, you are able to troubleshoot, research the root cause of problems, and thoroughly resolve defects. You provide excellent customer support. You take ownership of problems (even when outside your own domain), propose solutions, and either take ownership for their resolution or ensure a clear hand-off to the right owner. You participate in the interview process and help your team train SDE interns.

2.3 Moving to SDE II…

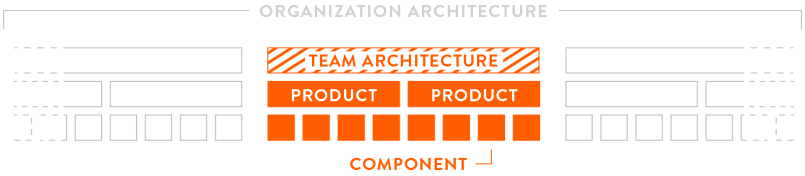
You will be considered for promotion to SDE II if you consistently demonstrate a combination of the following:

- You design, implement, and deploy software components and features. You solve difficult problems generating positive feedback.
- You have a solid understanding of design approaches (and how to best use them).
- You are able to work independently and with your team to deliver software successfully.
- Your work is consistently of a high quality (e.g., secure, **testable**[16], maintainable, low-defects, efficient, etc.) and incorporates best practices. Your team trusts your work.
- Your code reviews tend to be rapid and uneventful. You provide useful code reviews for changes submitted by others.
- You focus on operational excellence, constructively identifying problems and proposing solutions, taking on projects that improve your team's software, making it better and easier to maintain.
- You make improvements to your team's development and testing processes.
- You have established good working relationships with peers. You recognize discordant views and take part in constructive dialogue to resolve them.
- You are able to confidently train new team-mates about your customers, what your team's software does, how it is constructed, tested, operates, and how it fits into the bigger picture.

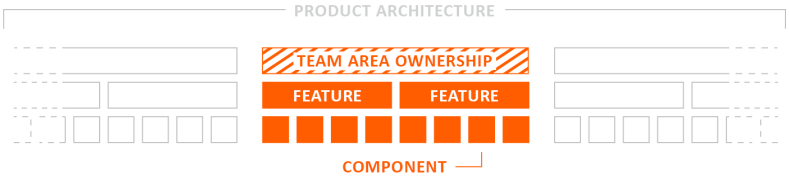
3. SDE II

3.1 Scope and Influence

SDE II scope of work is generally on a major portion of existing or new team software. This can include creating and modifying a large or significant set of components, a mid-size application, or service.



For teams that own major features or a technology that is part of a larger product architecture that is spread across multiple teams, SDE II scope may include creating and modifying a significant component or an entire feature in the portions of the technology that their team owns.



SDE II influence is still primarily within their team.

3.2 What you do…

You are a significant and autonomous contributor. Your work is consistently of high quality. You solve difficult problems, applying appropriate technologies and best practices. You work with your team to invent, design and build software that is stable and performant. You are proficient in a broad range of design approaches and know when it is appropriate to use them (and when it is not). Your solutions are pragmatic. You consider the legacy of the code you produce and write code that an SDE unfamiliar with the system can understand. You limit the use of short-term workarounds. You do things with the proper level of complexity the first time (or at least minimize incidental complexity). You create flexible software

[16] **Testable** – Creates automated unit tests where possible; manually tests legacy systems without automated testing support.

without over-engineering. You make appropriate trade-offs, re-use where possible, and are judicious about introducing dependencies. You are efficient with resource usage (e.g., system hardware, database, memory/CPU, etc.)

You work on project ideas with customers, stakeholders, and peers. You help balance customer requirements with team requirements. You help your team evolve by actively participating in the code review process, design discussions, team planning, and ticket/metric/COE reviews. You focus on operational excellence, constructively identifying problems and proposing solutions. You take on projects and make software enhancements that improve team software and processes. You work to resolve the root cause of complex problems, leaving software better and easier to maintain than when you found it. You are able to train new team-mates on how your team's software is constructed, how it operates, how secure it is, and how it fits into the bigger picture. You foster a constructive dialogue and seek resolutions in a professional way. You help recruit and interview for your team. You mentor and help to develop others.

3.3 Moving to SDE III…

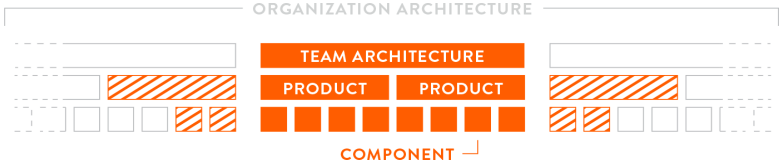
You will be considered for promotion to SDE III if you consistently demonstrate a combination of the following:

- You lead the design, implementation, and delivery of software in ambiguous and complex problem spaces (at the team level). Your efforts may produce new software, rework or even deprecate existing software. You heavily influence the design and write a significant portion of "critical-path" code. You make the right trade-offs. The problems you solve are complex, but your solutions are as simple as possible.
- You think in terms of architecture, not just code. You proactively work to improve the consistency and integration between your team's software and any related software (owned by other teams). You understand that many problems are not new and explore re-use or extending existing solutions first. You understand that creating new software is not always the right action.
- You influence your team's technical and business strategy by making insightful contributions to team priorities and approach. You take the lead in identifying and solving architecture deficiencies or areas where your team's software bottlenecks the innovation of other teams. You help your team deliver higher quality software faster (or fail faster to get to the right solution).
- You are able to communicate your ideas effectively to achieve the right outcome for your team and customer. You seek diverse perspectives, listen to feedback, and are willing to change direction if it creates a better outcome. You harmonize discordant views and lead the resolution of contentious issues (build consensus).
- You lead design reviews for your team and actively participate in design reviews of related software or other team software at your location.
- Your code, design, and implementation decisions set a great example to others and demonstrate best practices. You provide insightful code reviews and take ownership of outcome. ("You 'ship it', you own it.") You work very efficiently and routinely deliver the right things.
- You demonstrate your ability to influence, positively impacting the technical decisions made by one or two other teams (not your own). The influence can be via a collaborative software effort or by driving software best practices (e.g., security, quality, operational excellence, etc.).
- You actively participate in the hiring process as well as mentor others - improving their skills, their knowledge of your software, and their ability to get things done.

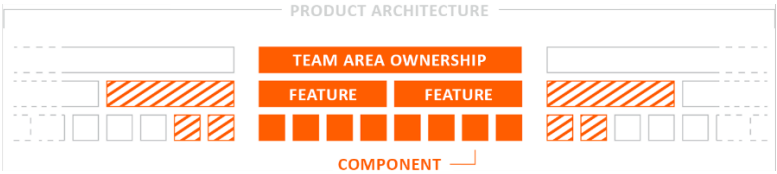
4. SDE III

4.1 Scope and Influence

SDE III scope is at the team architecture level. They may influence other teams, but only in the course of the work on their team's software. Again, two ways to visualize SDE III scope.



If the team owns technology that is part of a larger product architecture, SDE III scope may include taking the lead on the design and delivery of a major feature or they may personally create or re-architect a significant software component that other teams may depend on.



4.2 What you do…

You are considered a technical leader on your team. You work efficiently and routinely deliver the right things with limited guidance. Your work focuses on complex, ambiguous problem areas in existing or new software initiatives. You take the long term view. You consider where each system is at in its **lifecycle**[17] and where appropriate, proactively fix architecture deficiencies. You make existing systems simpler (e.g. by consolidating redundant systems, deleting unnecessary code, or removing/updating out of date documentation). You are able to take the lead on large projects that require the work of your *team*. You know how to divide a software project into parallel work that can be performed by you and other SDEs and then reassembled successfully into a cohesive launch.

You understand the business impact of your systems and show good judgment when making technical trade-offs between your team's short-term business or operational needs and long-term technology needs. You are a key influencer in team strategy. You drive mindful discussions with customers and peers. You bring perspective and provide context for current technology choices and guide future technology choices. You understand that not all problems are new (or require new software). You make appropriate architectural trade-offs (e.g., coarse or fine grained service separation?) Your code submissions and approach to work are exemplary – your solutions are inventive, secure, easily maintainable, appropriately scalable, and extensible. You write software that is easy for others to contribute to.

You take ownership of team architecture, providing a system-wide view and design guidance. You make things simpler. You drive engineering best practices (e.g., Operational Excellence, Security, Quality, etc.) and set standards. You work to resolve the root cause of endemic problems including areas where your team limits the innovation of other teams (bottlenecks). This may require you to influence software decisions made by *other* teams. When confronted with discordant views, you are able to find the best way forward and influence others to follow that path (build consensus). You actively recruit and help others leverage your expertise, by coaching and mentoring in your organization (or at your location). You provide technical assessments for promotions to SDE II and SDE III. You contribute to the professional development of colleagues, improving their technical knowledge and the engineering practices. You ensure your team is stronger because of your presence, but does not require your presence to be successful.

4.3 Moving to Principal…

You will be considered for promotion to Principal if you consistently demonstrate a combination of the following:

- You lead the design, implementation, and delivery of software in highly ambiguous and significantly complex problem spaces that have a long-term impact on a product, technology or architecture. Your efforts may produce new software, refactor, or even deprecate existing software. You heavily influence the design and write a significant portion of "critical-path" code. You make the right trade-offs. The problems you solve are significantly complex, but your solutions are as simple as possible.
- Your designs are noteworthy in some way (e.g., a significant refactor that simplifies or improves architectural quality, enables us to maintain a competitive advantage, offers significant extensibility, performance, scalability, or has organization-wide force multiplier or operational excellence impact). You only advocate for the creation of new software when it is necessary.

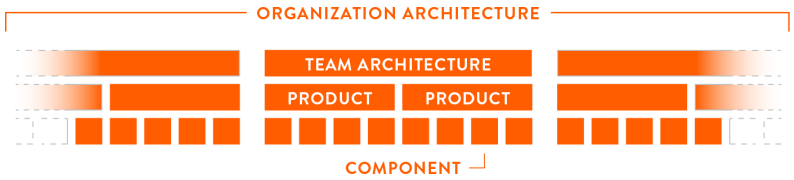
[17] **Lifecycle** extends from idea to design, to justification and approval, to build and launch, to the continued innovation of the system, to eventual migration to newer solutions and deprecation of older ones.

- You identify and tackle intrinsically hard problems. (e.g., major bottleneck or existing architectural complexity that slow down or prevent innovation, pose a critical business or security risk, undefined opportunity with potentially high impact, less prior art to be inspired by, etc.).
- You deliver artifacts that set the standard in your organization for engineering excellence, from designs to algorithms to implementations. Your personal code submissions and reviews of other people's code are instructive and make the overall code corpus better.
- You actively participate in design reviews, improving product delivery and aligning teams across your organization towards coherent architectural strategies. You bring clarity and simplicity to complexity, probe assumptions, illuminate pitfalls, and foster shared understanding.
- You are a pragmatic problem solver, applying judgment and experience to balance trade-offs between competing interests. You are flexible, adapting your approach to meet the needs of the team, project, and product.
- You actively recruit for Amazon as well as participate in the hiring/interview process.
- You play a significant role in the career development of others, actively mentoring and educating the larger engineering community on trends, technologies, and best practices.
- You keep abreast of industry trends. You effectively research and benchmark our technology against other competing software in the industry. You understand that many problems are not essentially new. You help others understand when it is appropriate to create new versus reuse or extend existing software solutions.
- You contribute to intellectual property (e.g., invent, submit patents, etc.)

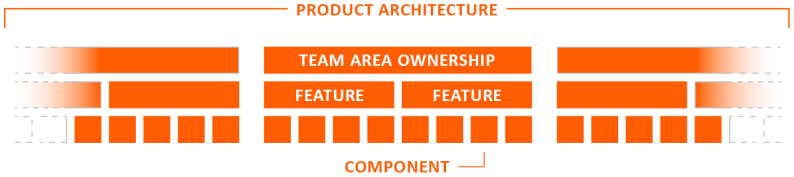
5. Principal SDE

5.1 Scope and Influence

Principal SDE scope varies by individual expertise. A Principal SDE may focus on a strategically important, significant, or critical piece of technology or they may be responsible for a large portion of organization architecture, which can include ownership of the design, quality, security, performance, availability, and operational aspects of the software being built by multiple teams.



If the Principal SDE is guiding a product architecture, they may take the lead on the design, or they may personally create or re-architect a significant software components that other teams depend on.



5.2 What you do…

You are a trusted part of the technical leadership of an organization, typically at the Director level. As a key influencer in planning strategy, you bring business and industry context to technology decisions. You set the standard for engineering excellence in your organization. Your architectures are exemplary in terms of efficiency, stability, extensibility, and the ability to evolve over time. You simplify processes and technologies. Your software is robust in the presence of failures, scalable, and cost-effective. Your coding practices are exemplary in terms of code organization, clarity, simplicity, error handling, and documentation. You tackle intrinsically hard problems, acquiring expertise as needed. You are able to decompose complex problems into straightforward solutions. You take the lead on projects that may require the work of several *teams* to implement. You are able to divide responsibilities so that each team can work independently and have the system come together into an integrated whole. You are flexible, adapting your approach to meet the needs of the

team, project, or product. You solicit differing views and are willing to change your mind as you learn more. You are adept at building consensus.

As a “hands-on” technical leader, you split your time between coding, design, and architecture based on where your skills will have the greatest impact (or in response to job requirements). For example, your expertise may be broadly applied – spread across many teams – involved in the technical strategy, design, and delivery of a significant portion of architecture. Or you may personally produce code for significant, critical, or demanding software and influence just the few teams close to it. The exact role you play may also change as a larger initiative progresses; during the early phases, broadly influencing several related teams and then subsequently spending weeks or months focusing on a particularly challenging system. Sometimes the mix goes the other way and you spend most of your time broadly influencing multiple teams while occasionally taking a deep dive into a critical, highly complex area in a particular team.

You amplify your impact by leading design reviews for complex software projects and/or critical features. You probe assumptions, illuminate pitfalls, and foster shared understanding. You align teams toward coherent architectural strategies. You educate, keeping the engineering community up to date on advanced technical issues, technologies, and trends. You participate, sharing knowledge and collaborating with other Senior Engineers, specifically attending and/or presenting at internal conferences, Principal Engineer community events and making yourself available to global developer outreach efforts. You help managers guide the career growth of their team members by mentoring, performing Principal promotion assessments, and participating in performance discussions.

5.3 Moving to Senior Principal - Tech

For “Moving to…” criteria, refer to the Sr. Principal – Tech Role Guideline.

SDE CLARIFICATION FAQ

1. Is there more guidance available on SDE expectations?

Yes! Check out:

- **Introduction to SDE Levels**
- **SDE 101** (for SDEs in their first 90 days)
- **Technical Interviewer Training**

2. What if we find a section of the SDE Role Guideline confusing?

Email sde-level-clarification@amazon.com. We consider the SDE Role Guideline a "living document" and will update it periodically based on feedback to improve its usefulness.

3. Is the SDE Role Guideline to be used for hiring?

Yes. Amazon role guidelines convey company-wide role and level expectations. To ensure we are consistent across all organizations and locations, it should be used in all aspects of assessing performance or level. This includes hiring, performance reviews, next-level goal setting, promotion justification and promotion reviews.

4. What is the SDE Technical Bar?

The Tech Bar for a level, includes any technical skills outlined in previous levels (i.e., a Principal SDE has all of the technical skills documented at lower levels). These are minimum requirements. Organizations may expect a higher Tech Bar, but not a lower one.

For Bar Raisers and Hiring Managers, here are specific SDE technical expectations by level:

SDE I:

- College education/personal study in Computer Science, Computer Engineering, Mathematics, Physics, or other STEM discipline. A degree is **not required** if the candidate has skills that meet the bar below.
- Able to code in one currently relevant, industry standard programming language (e.g., C, C++, C#, Java, JavaScript, Python, etc.)
- Can convert a design into code and deliver it using current software engineering best practices
- Writes secure, testable, maintainable code with minimal defects
- Understands a broad range of data structures and algorithms, and which to use, or not
- Knows how to test code. Can create a unit test and understands how to thoroughly test functionality.
- Able to classify, store, and handle data in accordance with policy
- Understands the maintenance characteristics, runtime properties, and dependencies of software; this includes operating system and build dependencies
- Able to identify software defects by reviewing code errors, logfiles, metrics, using debug functions, and other common troubleshooting techniques.

SDE II:

- Can design and build stable and performant software
- Able to write code that an SDE unfamiliar with their software can understand
- Knows a broad range of design approaches and when to use them (and when to not)
- Understands the legacy of coding decisions. Knows when a short-term workaround is a problem and when to limit this type of solution (because it may have serious consequences) and the implications of adding new dependencies.
- Knows how to be efficient with compute resources: hardware, database, memory/CPU, etc.
- Knows how to provide a code review. Provides useful coding reviews to others
- Able to constructively participate in technology design discussions, team planning, and metrics reviews

- Able to assess an existing software application and correctly identify where enhancements could be made to improve it (e.g., user experience, quality/ability to test efficiently, data handling, efficiency, performance, areas that make it risky to make code changes, maintenance requirements, etc.)
- Able to identify and resolve the root cause of software defects
- Able to instruct junior SDEs about software construction, operation, and security requirements

SDE III:

- Able to work on complex, ambiguous problem areas in existing or new software initiatives
- Can take the lead on a large software project, which may require the work of other SDEs on their team to implement
- Knows how to break software features down into components that can be developed in parallel work by them and other SDEs and then reassembled successfully into a cohesive feature launch
- Understands how to make technical trade-offs between short-term vs. long-term technology needs (e.g., business need vs. operations need or level of software quality or testability) and how to make architectural trade-offs (e.g., coarse or fine grained service separation?)
- Writes software that is easy for others to contribute to, is extensible, and maintainable
- Able to take responsibility for a team's architecture; provide a system-wide view and design guidance to the SDEs they work with
- Able to resolve the root causes of software architecture deficiencies (e.g., performance/scaling problems, too many dependencies, bottlenecks, significant latency or performance issues, data quality issues, etc.)

Principal SDE:

- Can design develop, and deliver a large software architecture. If they have depth expertise, they can design, develop, and deliver strategically significant or demanding software that changes how we engineer solutions
- Their architecture solutions are exemplary in terms of efficiency, stability, extensibility, and the ability to evolve over time. Their software is robust in the presence of failures, scalable, and cost-effective
- Their coding practices are exemplary in terms of code organization, clarity, simplicity, error handling, and documentation
- Able to decompose significantly complex engineering problems into straightforward solutions.
- Can take the lead on projects that may require the work of several teams to implement
- Able to take responsibility for the overall architecture of a major product or small organization, aligning the teams that own that software towards a coherent system strategy
- Can decompose a software architecture into components that can be owned and developed by different engineering teams. Able to communicate component divisions in a way that allows individual teams to work mostly independently. Current on advanced technical issues, technologies, and trends

5. Do we have examples for "Depth SDEs?

The "*Depth and Breadth*" concept is somewhat limited because it doesn't reflect how SDEs actually work and this becomes more evident at the Principal level. Most SDEs can go deep in one area or be broadly applied, depending on job requirements, their interests, or their own unique skills. However, some are expected to focus on a single strategically important technology, personally developing code for a significantly complex, strategically important, critical component or high-risk system. Others are expected to spend the majority of their time working at higher architecture levels, providing engineering leadership to several teams. There are also SDEs expected to simultaneously dive deep and broadly influence. All points on the spectrum are valid and acceptable.

Principal SDEs are given the discretion to split their time between code, design and architecture levels to achieve the highest benefit and impact. If they work deeply at the code level, it would be expected that the code produced would be highly leveraged, business critical, industry leading, etc.

6. How long do SDEs need to demonstrate the next level to be considered for promotion?

Every individual promotion case is unique and there are no strict guidelines. Essentially the SDE up for promotion must be *consistently* demonstrating a combination of the required "Moving to" next level criteria, our Leadership Principles, and/or have other evidence that they will ultimately be successful at the new level.

7. Does an employee have to work towards the next level?

No. Any employee that consistently delivers results and demonstrates they are continuing to meet Amazon's high bar for their role has value. However the bar is always rising. When an employee is no longer meeting the bar for their role, a candid conversation about their performance and career aspiration is in order. It may be that a different role is a better fit for their skills and interests.

8. How much cross-team influence do SDE need to be considered for promotion?

The answer depends on the SDE level. In general (and this is consistent with this guideline):

- **SDE I into SDE II** – No requirement to work with/influence other teams.
- **SDE II into SDE III** – Needs some experience working with/influencing other teams. To be promoted, the manager ideally has two examples of how the SDE influenced teams other than their own. The ideal scenario is how they negotiate the design or technical decisions made in a project situation where their team's software is related to another team's software in the architecture. Another scenario might be SDEs navigating technical decisions with hardware engineers on devices or mechanical systems. If the SDE is in a small development center or on a secret project, they might demonstrate influence driving best practices, leading design reviews, and/or offering technical learning sessions at their location.
- **SDE III into Principal** – Requires significant experience working with and influencing multiple teams.

9. Is tenure a factor in promotion readiness?

No. A manager needs Amazon examples and artifacts to justify readiness for the next level. They also either need experience or enough feedback about the SDE to assess areas where they are still growing. If hired into a level, a manager can also refer to the SDE's interview feedback, paying specific attention to any Bar Raiser or interviewer concerns. This can provide a way of focusing goals to demonstrate the next level and structure a promotion justification.

10. How do SDE promotions work in Agile teams where everyone contributes to the design?

This is relevant to SDE I to SDE II and SDE II to SDE III promotions. It is less relevant at the Principal level because those projects often span teams and initiatives have a longer time horizon.

- **SDE I into SDE II**: The expectation for SDE II is that they understand a broad range of design patterns and know which to use (and which not). The manager needs examples for various situations, so they can:
    - As part of a sprint, assign the SDE the task of proposing a design for a component or feature. They should elaborate on why they chose that approach and not another. Then let the team dissect it. How often are they correctly selecting the right approach? Can they defend it?
    - Assess how the SDE contributes in team discussions where other SDEs are proposing a design approach. How constructive is their feedback? Do they offer alternative ideas?
- **SDE II into SDE III**: The expectation for SDE III is that they can take the lead and design the right approach for a major software effort. The manager needs to assess (or make sure they have more senior SDEs assess) whether the SDE's design was appropriately simple and if their approach is correct (e.g., solution exists, but it needs work and would create a dependency vs. team creates their own solution which requires longer time to launch and adds to operations load, etc.) Some ideas:
    - The SDE could assess their team's software from a long-term perspective and propose projects that are needed to keep the system evolvable, relevant, and/or to take advantage of new opportunities. They make the case, propose the design, and influence the manager that it's the right course to take. If approved, they take the lead on getting their design reviewed, as well as recommending how to split that work into components and/or features (stories) so that others can work in parallel. They take the sprint tasks for the critical components or most high risk features.

- The manager could assign the SDE their major project. The SDE would recommend the design approach, which includes how to make sure they create the best customer experience, test to validate quality, and maintenance/operations plans. They lead the discussions with the team to give them the opportunity to weigh in, drive the design review with more senior SDEs, and then recommend how to split that work into stories so that others can work in parallel. They take the sprint tasks for the critical components or most high risk features.

When the manager has several SDEs interested in working towards next levels, they would need to carve out areas of focus to allow them to lead in parallel. When a team doesn't have that ability, it may be necessary to recommend moving to another team within their organization that has the right level of projects to grow the SDE.

11. Are there examples of the types of SDE projects at each level?

No. The goal of an Amazon Role Guideline is to describe *company-wide* expectations. Amazon has a huge variety of technologies, customer needs, and organizational structures. And as we all know, Amazon is not prescriptive. It is expected that SDEs and their management teams understand their space and how to map the generalized level guidelines to appropriate goals in their team.

Another reason that we do not include specific examples is to avoid anchor[18] bias. By providing a reference for each level, we risk teams overly indexing on the listed types of projects. They may attempt to pattern match rather than applying more critical thought on whether an employee is meeting the criteria to be considered for promotion. Thus, we avoid citing artifacts that could be used as inappropriate barriers. ("*The project described in this promo doc is similar to project foo on the list of projects that got people promoted, but this particular project is missing N things that foo had, therefore, I'm not inclined to promote*.")

The projects that SDEs work on are only one significant part of their performance evaluation. Leadership, ownership and technical excellence are important SDE skills in addition to a track record of project delivery.

[18] To understand further, see: Anchoring