Software Engineer Career Schema

This document describes the career levels and their associated requirements for engineers. It is developed with reference to the career schemes of top engineering companies to allow us to remain relevant and competitive. For the general theory of leveling, performance evaluation, and promotion processes refer to the Levelling, Performance, & Promotions guide.

There are 3 main goals for providing explicit level requirements:

- Ensure that the strongest contributors are given the most authority
- Provide clear personal development targets
- Create a measure of fairness and equity

Each level's requirements are intended to be clear, testable requirements to promote someone to that level. A person should not be able to satisfy the requirements for a level just by virtue of having been assigned to it. They are deliberately written to describe **demonstrated ability** and **not assigned job scope**. At low levels, they also reflect the minimum unit of work one can trust someone to complete without explicit supervision.

There are no salary bands for each level. Instead each specific level has a fixed salary, but there are multiple levels per semantic tier. The goal is to have compensation purely reflect an individual's level of contribution. We want to avoid compounding factors such as contract negotiations, years in the organization, and general favour from the leadership. Having explicit discrete sublevels makes it clearer that increased compensation is based on demonstrated progress towards the next major level of contribution.

Levels

Software Developer - Aptitude for engineering but lacks some basic skills

- Software Developer 1 - Entry point for students or mid-career switchers who have completed basic coding courses
- Software Developer 2

Associate Software Engineer - Entry level trainees

- Associate Software Engineer 1 - Entry point for most fresh CS graduates
- Associate Software Engineer 2
- Associate Software Engineer 3

Software Engineer - Competent individual contributors

- Software Engineer 1
- Software Engineer 2
- Software Engineer 3

Senior Software Engineer - Fully proficient individual contributors who can provide technical leadership

- Senior Software Engineer 1 - Resting level for most engineers
- Senior Software Engineer 2
- Senior Software Engineer 3

Split between individual contributors and managers. High level engineers are in charge of engineering itself. Engineering managers are in charge of the people who work as engineers.

Staff Engineer - Champions for engineering progress
Engineering Manager - Fully proficient engineers who also competent people managers

- Staff Engineer 1 / Engineering Manager 1
- Staff Engineer 2 / Engineering Manager 2
- Staff Engineer 3 / Engineering Manager 3

Principal Engineer - Paragons of critical engineering leadership
Engineering Director - Managers who are proficient enough to manage other managers

- Principal Engineer 1 / Engineering Director 1
- Principal Engineer 2 / Engineering Director 2
- Principal Engineer 3 / Engineering Director 3

Individual Contributors

Software Developer

Software Developers are people who **have an aptitude for engineering but lack some basic skills** to be productive as an engineer. They have the ability to solve coding problems, but are lacking in areas such as fluency in their programming language, debugging techniques, or an understanding of algorithms. Overall, they are able to complete basic tasks with heavy guidance but without being a net negative on the team's productivity.

Concretely this means Software Developers are able to:

- Solve computational thinking problems
- Write code to implement a given well defined algorithm
- Understand and discuss computer science concepts

Associate Software Engineer

Associate Software Engineers are **entry level engineers**. They have demonstrated an aptitude for computer science and software engineering even if they are not yet familiar with the tools and processes of working in a production environment. Overall, they are able to complete assigned engineering tasks with some guidance. They achieve some impact through working on product features.

Concretely this means Associate Software Engineers are able to :

- Write code fluently in at least one programming language
- Breakdown a simple requirement into computational steps
- Debug their code and lookup necessary information when problems occur
- Correct their approach when they have recognized problems with it
- Reason about algorithmic correctness, their use, and their runtime performance
- Communicate clearly about technical topics

Impact:

- Incremental improvements to features

Citizenship:

- Assist with onboarding of new software engineers onto the team
- Conduct interviews of software engineering candidates with guidance
- Lead a talk at OGP team sharing

Software Engineers

Software Engineers are **competent individual contributors**. They are comfortable with engineering tools such as source control, error monitoring, automated testing, and more. They can successfully run systems in production, though they may be unable to design such systems themselves. They can reason about the practical implications of a system design and can make useful contributions to design discussions. Overall, they are able to prioritize engineering tasks for themselves and complete them independently.

Concretely this means Software Engineers are able to:

- Write code that matches the readability and design standards of the team
- Implement systems from a given architectural design
- Understand the design goals and limits of a given system and work around them
- Prioritize engineering tasks effectively and avoid getting stuck on low impact work
- Use engineering tools effectively
    - Collaborate with other engineers by writing well defined pull requests
    - Participate productively in a code review both as a reviewee and reviewer
    - Branching and merging appropriately in source control
    - Configure build tools for simplified deployment and development
    - Setup automated testing to prevent code regressions
- Operate production systems reliably
    - Setup and operate cloud infrastructure for a given architectural design
    - Implement logging and be comfortable searching through logs
    - Configure basic alert systems to minimize downtime
    - Deploy code to production using practices that minimize risk of user interruption
    - Respond to production outages and recover from simple errors
    - Conduct post mortems detailing the significant events and root cause analysis

Impact:

- Propose and drive new features independently
- Coordinate with team members to drive new features

Citizenship:

- Independently onboard new software engineers onto the team
- Independently conduct interviews of software engineering candidates
- Lead a talk with an audience outside OGP (e.g. talk at GovTech, WeWork, etc)

Senior Software Engineers

Senior Software Engineers are **fully proficient individual contributors** who are able to provide technical leadership to their teams. They have implemented a variety of systems and understand the tradeoffs between different databases, frontend frameworks, and other system design choices. They can independently identify the important characteristics of the system, devise the necessary protocols and infrastructure, and choose the appropriate technologies and services. They are able to set up a skeletal framework for the project so that other engineers can easily build upon it. Overall, when given a project they are able to devise the architecture, break it down into engineering tasks, and make rapid progress on those tasks.

Concretely this means Senior Software Engineers are able to:

- Establish the coding conventions for a project
- Identify important system characteristics such as idempotency, data durability, or privacy given a general project description
- Devise engineering solutions to bureaucratic requirements and constraints
- Design the data model for the system to both be elegant but sufficiently extensible
- Design the infrastructure to be fast, resilient, secure, and efficient
- Design the system protocols to provide the desired semantics
- Design the code modularization to be easy to understand and maintainable
- Communicate these designs clearly to the team to discuss and understand
- Prioritize tasks for junior team members and provide guidance to help complete them
- Represent the team in technical discussions with other teams
- Setup the core infrastructure, initial codebase, and build tools to allow teammates to start making contributions efficiently in a well defined framework
- Make prolific contributions to the codebase themselves
- Triage, diagnose, and recover from complex production outages

Impact:

- Drive engineering for a new product
- Mobilise 3-5 engineers to work on product
- Drive adoption of new framework/approach that increases productivity of engineers on the team
- Assist with recruitment efforts by attracting external software engineering talent to apply to OGP, through talks or networks

Citizenship:

- Guide more junior engineers to achieve goals more efficiently/effectively
- Independently conduct interviews of core/senior software engineering candidates
- Lead a talk / Evangelise OGP engineering with broader engineering community (e.g. Talk.JS session)

Senior Software Engineer 1 is the resting level for engineers. Progression beyond this point is not expected and is only done in the case of exceptional individual proficiency or organizational impact. This is the level which most engineers can comfortably remain through to their retirement.

Senior Software Engineers can also start having a small number of direct reports and developing as managers. See the management section for more details

Staff Engineers

Staff Engineers are **champions for engineering progress**. In addition to being prolific individual contributors, they proactively identify, initiate, and coordinate worthwhile engineering projects for the organization to pursue. They have deep expertise in at least one technical domain. They can critically evaluate existing system architectures, develop aspirational goals for the codebase, and plan a feasible path to get there. Additionally, Staff Engineers are comfortable working with people across the organization. They can communicate their engineering goals clearly, persuade other engineers of its merits, and coordinate its smooth implementation across the codebase. Overall, they independently push substantial engineering projects to be successfully implemented through their open ended technical leadership.

Concretely this means Staff Engineers are able to:

- Understand and operationalize new academic research in a technical field
- Discern the reasons why an existing system was built in a particular way and what limits it is imposing on future development
- Design an improved structure for an existing system that addresses these limits and draft an implementation plan to get there smoothly and efficiently
- Identify high value engineering projects and technical debt problems
- Persuade other engineers and stakeholders of the importance of these opportunities and to devote resources to them
- Coordinate implementation progress across multiple teams to allow for a smooth transition

Impact:

- Drive engineering for multiple products
- Mobilise 6-12 engineers to work on multiple products
- Establish new framework/approach that increases productivity of engineers on the team
- Assist with recruitment efforts by attracting external senior software engineering talent to apply to OGP, through talks or networks

Citizenship:

- Guide senior engineers on principles of system architecture
- Improve interviewing processes for software engineering candidates to be more accurate and precise
- Lead a talk at industry-leading conferences (e.g. STACK)

Principal Engineers

Principal Engineers are **paragons of engineering and are the critical engineering leadership** of the organization. They create the long term vision for what the organization should be building towards and structure the groundwork for building projects to get there. They hold indispensable engineering expertise for the organization's core mission and are actively pushing the boundaries of their technical domain. This expertise is the organization's core value proposition, and the basis for coordinating the work of the entire engineering team.

TODO: Concrete benchmarks for Principal Engineers

Management

The engineering management level requirements are differentiated using similar semantics to the individual contributor level requirements.

For individual contributors:

- Associate Engineers are still in training
- Intermediate Engineers are competent individual contributors
- Senior Engineers are fully proficient

For those pursuing engineering management:

- Senior Engineers can start gaining some experience having direct reports
- Engineering Managers are competent at people management and running a team
- Engineering Directors are fully proficient and proactively improve the organization

Senior

In addition to being fully proficient individual contributors, some Senior Engineers can manage a small team. In this capacity, they train their reports while working alongside them. Explicitly teaching them engineering skills, explaining technical design considerations, and guiding them through unfamiliar tasks. They assign prioritized tasks to their team and track their progress in completing them. They communicate with their reports helping address both technical questions and career development concerns. Overall, they provide mentorship and tactical leadership to junior engineers.

Concretely, this means in addition to their individual contributions, Senior Engineers who take on a management role are able to:

- Keep a project's tasks and progress well organized
- Communicate comfortably and build rapport with other engineers
- Critically assess the performance of other engineers
- Teach engineering concepts and skills effectively

Engineering Managers

Engineering Managers are fully proficient engineers who are also competent people managers. Instead of making technical contributions themselves, they focus on supporting other engineers and help them work more effectively. Their role is not just to manage the engineers assigned to them, but to proactively find new engineers and convince them that they are worth working for. They build the organization by recruiting and interviewing new people to join. They organize training and mentorship to build the competencies and careers of the engineers who report to them. They design engineering processes to allow the team to collaborate efficiently while reducing production failures. They breakdown projects into manageable pieces of work and allocate them considering both the project needs and the development opportunities for their team. Overall, develop strong engineering teams, make sure they are well taken care of, and are making strong contributions to the organization.

Concretely this means Engineering Managers are able to:

- Identify strong candidates, reach out to them, and convince them to join the team
- Develop the capabilities of their engineers by providing effective mentorship and training opportunities
- Provide clear and actionable feedback to their reports
- Communicate clearly regarding project goals, career development opportunities, corporate systems, and team culture
- Allocate engineers to projects while considering task priority, skill requirements, and development opportunities
- Keep the team aligned using project management systems to track task progress and recalibrating as necessary
- Identify bottlenecks and roadblocks in engineering processes and clear them

Engineering Directors

Engineering Directors are fully proficient engineering managers. The critical capability is that they are able to effectively manage other managers. They have the experience to understand the tradeoffs between a variety of approaches to performance evaluation, career development, engineering workflows, team culture, and the many other aspects of management. They can identify the key characteristics of the organization's operating environment, design the appropriate structures and processes, and guide their implementation. Though they are unlikely to still code themselves, they have deep technical knowledge that allows them to debug complicated systems and allocate resources to unblock critical engineering work. They are strong communicators; Building alignment across the engineering organization, as well as explaining the importance of complicated technical topics to lay audiences. Overall, they provide open ended organizational leadership by building the core corporate structures for other managers and engineers to operate within.

TODO: Concrete benchmarks for Engineering Directors

Annex 1: Theory

There are 3 main goals for providing explicit level requirements.

- Ensure that the strongest contributors are given the most authority. This allows the organization to make better decisions overall.

- Provide clear personal development targets for members of the organization. This pushes them to learn the right skills, work on the right projects, and make the right decisions on those projects. Poorly written level requirements create perverse incentives that cause dysfunction across the organization.

- Create a measure of fairness and equity across the organization. Having the wrong people promoted for opaque reasons is extremely demoralizing, so it is important that the right people get promoted for clear reasons.

For example, “manage a team of 10 engineers” or “works with teams across the organization” are poor requirements because any person given that role would automatically qualify. This would allow unqualified people to squat at positions using a self-fulfilling prophecy. This degrades decision making in the organization, demoralizes more qualified individuals, and prevents organizational growth.

Using adjectives to differentiate between levels is discouraged but unavoidable, and so should at least try to be clearly distinguishable. For example, having successive level requirements that are “writes code well”, “writes code masterfully” and “writes codes exceptionally” is bad because it is extremely difficult to judge between them. In contrast, having a single level requirement for “clear concise code” still necessitates making a judgement, but is theoretically much easier to keep consistent.

While objectivity is desired in level requirements, it is important to exclude objective but immaterial measures. Measures such a “8 years of experience”, “published internationally”, or “managed deals worth $10 million” are proxies for an individual's qualification for the role, but are themselves not directly relevant. Generally, it is preferable to identify the actual desired qualities such as “can code fluently in Java”, “understands the latest developments in the field”, or “able to negotiate deals and allocate funds in an efficient manner”. While the use of proxy requirements is sometimes unavoidable as the underlying desired traits are hard to assess, the problem occurs when objectivity is pursued at the cost of relevance. While more objective measures in themselves are desirable, it is usually necessary to retain some judgement based measures to avoid systematic irrelevance.

Years of experience in particular should be avoided. It is very tempting as a proxy for someone’s capability and in the short term is very closely associated with that. However, it specifically hinders the most talented individuals while benefiting those who have the fewest alternative options. In the long term, this results in the systematic loss of the most capable people as they pursue opportunities elsewhere. This lowers the average ability and increasing the average years of experience associated with a level of capability. This cycle repeats, resulting in an organization whose main distinguishing characteristic is that everyone there takes a long time to learn to do not very much at all.

Given these requirements, each level semantic is broken into 2 segments. The first is a paragraph of illustrative prose. It is optimized to build an intuitive narrative about what people of that level are like, but may not provide useful benchmarks to evaluate an actual promotion.

Concrete benchmarks are provided in the second segment as a list of specific capabilities a person should have at that level. The goal is to provide an unbiased list of checks that can separate the people who are actually qualified from those who may just fit our inherent biases. While they can be hard to distinguish sometimes, these are deliberately written as capabilities rather than responsibilities of that level. Responsibilities are tautologically given to someone of that level, while someone lacking capabilities will still be unable to meet these benchmarks.

Together, these 2 segments create a stable but encompassing requirement for a level. The benchmarks provide the anchor points, while the narrative fills in the expectations for anything non specifically defined. This reduces semantic drift while still giving space for judgement to be exercised.

Structure

The engineer career schema is divided into major semantic tiers, each of which are subdivided into 3 levels. This allows for clear and stable benchmarks of what is required at each stage of a career, while still accommodating the continuous spectrum of performance.

Semantic tiers are career milestones. These are checkpoints in an engineer's career. However, in order to have clear semantic tiers the distance between these tiers is necessarily large. This means that within a particular semantic tier there will be people across that range of contribution. In order to equitably accommodate these variances, each semantic tier is broken down into multiple actual career levels.

The written requirements for a particular tier are the requirements for the first level of that tier. So someone fulfilling the requirements for a Senior Engineer would be a Senior Engineer 1. As individuals make progress towards the next tier, they can be moved up to the next level in their current tier. As the requirements cover a large set of skills, individuals will likely make progress at different rates in different areas. This means that mid tier levels will not have their own written requirements. The main requirement being that all individuals within the same level are at similar levels of contribution and display roughly the commensurate fraction of characteristics from the next semantic tier.

There are no salary bands for each level. This system of multiple levels per semantic tier is a substitute for having levels with salary bands. At most organizations, there are requirements for every level but allow salaries within each level to vary. While theoretically this could be used purely to account for varying levels of contribution, in practice it is compounded with a variety of factors such as contract negotiations, years in the organization, and general favour from the leadership. Having explicit levels at least theoretically provides some structural clarity that levels and compensation are purely dependent on an individual's level of contribution.

Firstly, this creates a more trusting equitable environment. People who are at similar levels of contribution will have the same level and thus the same level of compensation. It removes any minor pettiness or envy that minor salary deviations can create.

Secondly, it removes the incentive for wasted effort on salary negotiations. In places where employees can have varying compensation, the selfish incentive for each employee is to spend time and resources negotiating their offer and pursuing alternate offers. Employees who are willing to engage in this unproductive adversarial behavior are rewarded. In the long run, this results in productive time being wasted, higher stress levels as employees feel the need to fight for their compensation, and lower levels of trust overall due to inequitable outcomes. With fixed salaries at each level, it aligns the incentives to promotion. The best way to get paid more is to do your job well and get promoted. Of course, this requires that the promotion process itself is well functioning.

All the problems with salary bands could theoretically equally apply to the levelling system. Tenure based promotions are the norm at many places and employees may seek competing offers to try and leverage a promotion. However, this at least eliminates salary variation as a source of dysfunction, allowing more focus to be placed on guarding the sanctity of the levelling process. And unlike the salary negotiation process, the levelling process can be clear and visible to minimize the likelihood of subversion.

Leadership

Higher level engineers are expected to exhibit some degree of leadership even as individual contributors. High level individual contributors will have at most 4 direct reports, and often have none. While engineering managers manage engineers, high level engineers lead engineering

If an engineer progresses beyond the Senior level, they can either remain as an individually contributing Staff Engineer, or switch over to a management role as an Engineering Manager. Both roles require a high engineering proficiency. However, while Staff and Principal Engineers focus on tackling bigger engineering problems, Engineering Managers and Directors instead focus on recruiting, training, and organizing engineers to work efficiently.

Starting from the Senior level, individual contributors can have a small number (<5) of direct reports even as their main focus is still on their individual contributions. Engineering Managers may occasionally help fix minor issues, but they will have a large number (>9) of direct reports and their focus is on developing their team.

Comparing engineering managers vs high level individual contributors:

- Engineering managers are in charge of the people who work as engineers. They are responsible for actively recruiting engineers to the organization and convincing them to join their team. They are responsible for tracking, enabling, and developing the skills and careers of engineers reporting to them. They allocate engineers across projects. And they improve and fix problems in engineering processes. They may not code themselves, but retain a strong technical background and ability to reason and make critical technical decisions. They are ultimately responsible for making sure the engineers themselves are well developed, cared for, and utilized.

- High level individual contributor engineers are in charge of the work of engineering itself. They are paragons of engineering, still being prolific contributors to the codebase. They build key infrastructure that makes it easy for other engineers to build onto. They provide technical guidance and thought leadership, helping junior engineers learn and implement systems more effectively. They initiate new projects and push the organization to work on important engineering problems with higher standards. They are ultimately responsible for making sure the engineering output is impactful, maintainable, and efficient.

In theory, principal engineers determine what we should be building, while engineering directors determine how to organize the engineers to build it. This is in contrast to traditional organizations where non-technical management provides technical direction, but leaves execution up to technical staff. In practice, because the question of what should be built and what can be built are so intertwined, both high level individual contributors and managers will need to closely collaborate to determine the right path forward.

Note that in contrast to most traditional career paths, most engineers will not become engineering managers during their career. This is an intended design as being a good manager requires specific aptitudes and skills that are not reasonable to expect of every person.