IT Project Guidance

Resourcing IT Developers and Architects

Version:

0.2

## Purpose

This document provides guidance on selecting appropriate developer resources for IT projects. It highlights the distinctions between different types of developers, ensuring that organisations can make informed decisions when assigning responsibilities. By understanding these roles, organisations can reduce misallocation of resources, minimise project risks, and improve overall software quality and maintainability.

## Synopsis

While all developers engage with code to some level, their skills, expertise, and focus areas vary significantly. Some specialise in maintaining or extending existing systems, while others focus on integration, platform development, deployment automation, or testing. Only a small subset are system developers, capable of designing and building software from the ground up. This document categorises these roles, identifies common misconceptions in resourcing, and outlines the risks of mismatching developers to project needs. It also examines the implications of architectural decisions, including platform selection and system design, ensuring that resourcing strategies align with long-term organisational goals.

## Contents

[Purpose 1](#_Toc190522638)

[Synopsis 1](#_Toc190522639)

[Contents 2](#_Toc190522640)

[Introduction 3](#_Toc190522641)

[Developer Roles 3](#_Toc190522642)

[Support Developers 4](#_Toc190522643)

[Integration Developers 4](#_Toc190522644)

[Platform Developers 5](#_Toc190522645)

[Functional Consultants 5](#_Toc190522646)

[Deployment System Developers 5](#_Toc190522647)

[Test Developers 6](#_Toc190522648)

[System Database Developers 7](#_Toc190522649)

[Front End Developers 8](#_Toc190522650)

[System Developers 8](#_Toc190522651)

[Full-stack Developers 9](#_Toc190522652)

[Inconsistencies of Terms 9](#_Toc190522653)

[Inconsistencies of Capability 10](#_Toc190522654)

[Risks 11](#_Toc190522655)

[Conclusion 12](#_Toc190522656)

[Appendices 13](#_Toc190522657)

[Appendix A - Document Information 13](#_Toc190522658)

[Versions 13](#_Toc190522659)

[Images 13](#_Toc190522660)

[Tables 13](#_Toc190522661)

[References 13](#_Toc190522662)

[Review Distribution 13](#_Toc190522663)

[Audience 14](#_Toc190522664)

[Structure 14](#_Toc190522665)

[Diagrams 14](#_Toc190522666)

[Standards 14](#_Toc190522667)

[Acronyms 14](#_Toc190522668)

[Terms 14](#_Toc190522669)

## Introduction

While all developers write code, their skills, experience, and focus areas differ significantly. Recognising these distinctions allows organisations to allocate resources more effectively, reducing misspending and minimising the risk of project failure. Selecting the right type of developer ensures that key project constraints—availability, cost, and capability—are appropriately managed.

A misconception in IT project resourcing is the assumption that all developers code, therefore can successfully design and build a system from scratch, or that all developers are interchangeable when it comes to deployment and integration.

However, whole system development is distinct from developing platform-specific extensions, initial deployment, ongoing maintenance, or system integration. It is rare to find resources that have experience at more than one, let alone all tasks involved, and even fewer who are good at them.

This document outlines the key categories of developers, their roles, and the risks associated with mismatching resources to project needs.

## Developer Roles

The following is a high-level categorisation of the most common types of developer specialisations. While all the following roles involve “coding” to some level, their focus, expertise, and responsibilities vary significantly.

Additionally, it is common for individuals within each specialisation to refer to themselves simply as "developers" in their day-to-day work, which can sometimes obscure the critical distinctions between their roles. Understanding these differences is essential for correctly resourcing IT projects and ensuring that developers are assigned tasks that align with their capabilities.

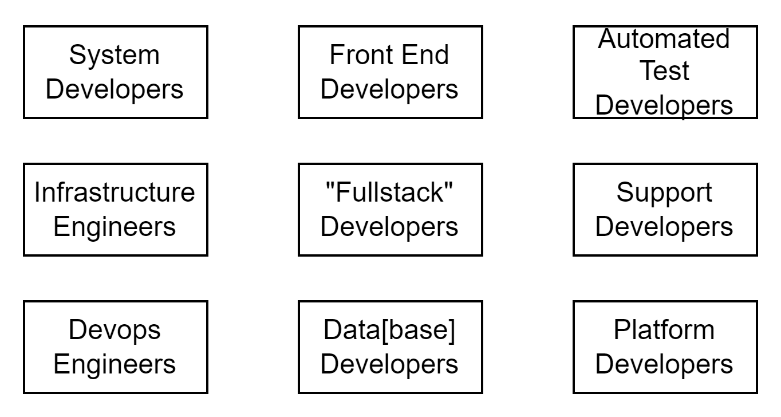


Figure 1: Common developer categories

Note:  
There are more specialised and less common developer categories; however, they have been excluded from this document for the sake of simplicity.

### Support Developers

Support developers primarily maintain and extend existing systems rather than build new functionality. Their role requires adaptability, as they often rotate across multiple systems, some of which may be legacy applications with outdated architectures. Their core responsibility is ensuring system stability, diagnosing issues, and making targeted fixes or small-scale modifications.

Their work is inherently risk-averse—focused on resolving specific issues with minimal disruption rather than implementing large-scale changes. Stability is their primary concern, and they typically avoid unnecessary changes unless required to resolve issues or maintain compatibility. While they are typically proficient in debugging and troubleshooting, they are not expected to have the breadth of experience required to design and build new systems from the ground up.

Furthermore, their exposure is primarily to systems that are already broken or deficient in execution, scope, or completeness. In essence, their experience is shaped by working gingerly around imperfect systems rather than having a mandate to imagine and create optimal ones.

### Integration Developers



Figure 2: Integration Development

Integration developers specialise in enabling interoperability between systems built by different teams or organisations. Their expertise lies in working with APIs, middleware, data transformation tools, and integration patterns rather than developing standalone applications.

They ensure seamless communication between disparate systems but do not typically modify core application logic. Their work focuses on facilitating smooth data exchange, handling authentication between systems, and ensuring that services interact as expected. One could summarise their role as dealing with outlets and plugs—adapting and reshaping data to fit between systems rather than designing the systems themselves.

### Platform Developers

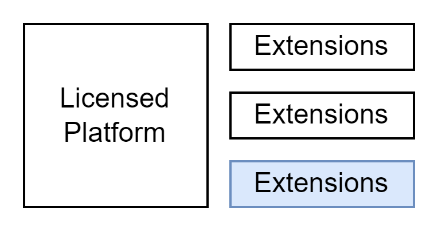


Figure 3: Platform Development

Platform developers extend the capabilities of existing platforms by building applets, plugins, or extensions within predefined extension points. Unlike system developers, they do not have access to or the ability to modify core system code; instead, they leverage existing APIs and frameworks to enhance functionality within the constraints of the underlying system.

Their expertise lies in optimising and extending platforms rather than designing new architectures. They excel at working within vendor-defined limits, ensuring compatibility with system updates and security policies, enhancing visible functionality without altering the underlying structure. Most are not trained or experienced in system architecture or ground-up development, making them adept at adding a customer specific polish—like frosting on a cake—while leaving the fundamental system design unchanged.

#### Functional Consultants

Related to platform developers are Functional Consultants, a term that has gained broader industry use with the rise of SaaS and enterprise platforms like Salesforce, Microsoft Dynamics, and SAP. With strong customer interaction skills they act as intermediaries between business requirements and technical implementation, combining a deep knowledge of the platform’s configuration limits with an ability to work with platform developers to leverage its capabilities to align with business goal. Like platform developers, they are not system developers themselves, as they operate within the boundaries of an existing ecosystem rather than designing core application logic from scratch.

### Deployment System Developers

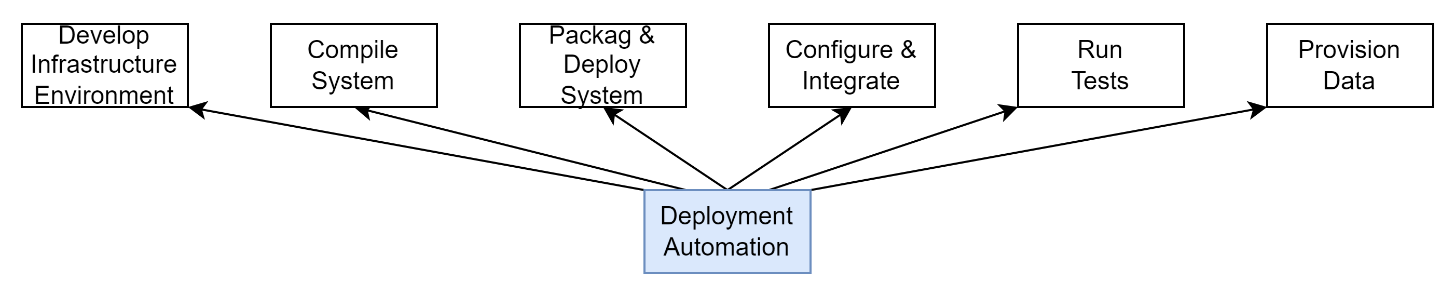


Figure 4: Deployment Pipeline Development

Deployment system developers specialise in automating software deployment “pipelines” to ensure efficient and reliable software delivery. Their work overlaps with DevOps practices but remains distinct—while DevOps covers broader operational concerns, deployment system developers focus specifically on automating software releases, managing infrastructure as code, and ensuring continuous integration and deployment (CI/CD).

Their role is crucial in maintaining deployment consistency, reducing manual intervention, and ensuring smooth transitions from development to production. However, their expertise does not typically extend to support, platform, integration, or system architecture and feature development. Their skill set is best summarised as creating and maintaining the factory in which other developers work. They are essential but are not system developers in their own right.

Their work might be passed on to be maintained by either system support developers or if the organisation has them deployment support specialists.

### Test Developers

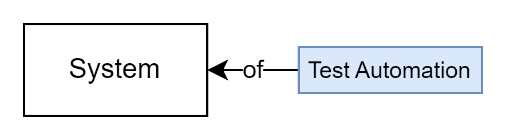


Figure 5: Test automation development

Test developers create automated test suites according to the specifications set by test analysts. Their primary focus is on writing code that ensures software behaves as expected under different conditions. While their work involves coding, their approach is distinct from system development, as it prioritises verification of contractual obligations rather than feature creation.

They can be likened to food testers rather than chefs, ensuring quality but not creating the product itself. While their absence may not be immediately felt during the development phase, the lack of automated testing later increases operational costs and slows maintenance, iteration, and innovation. Without automated tests to ensure quality is maintained, release cadence is constrained to the speed of humans (months) rather than machines (seconds).

### System Database Developers

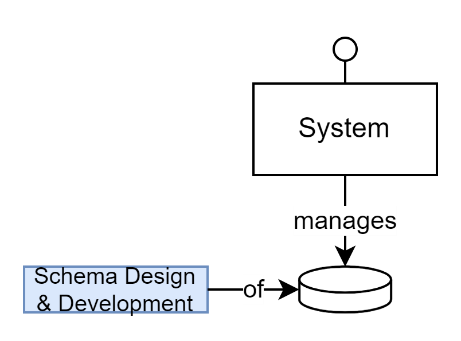


Figure 6: Database Development

Database developers remain essential specialists in certain types of development, particularly in data warehouse design, data flow engineering, and large-scale analytics systems. These areas require expertise in ETL processes, data modelling, indexing strategies, and query optimisation—skills that are not typically the focus of system developers. Their role is critical in ensuring that complex data solutions are scalable, efficient, and optimised for long-term maintainability.

While advancements in Code First approaches, ORMs, and improved database tooling have enabled system developers to handle many relational database needs independently, database specialists continue to provide valuable expertise. Their deep understanding of data structures, query optimisation, and storage strategies allows them to anticipate and mitigate performance risks that may not be immediately evident in application development. Additionally, they play a key role in ensuring data privacy, compliance, and governance, particularly in cross-organisational data-sharing scenarios.

The rise of polyglot persistence—where different types of databases (NoSQL, graph, time-series) are used for specific purposes—has broadened the landscape of data storage. While system developers can work with a variety of database models, database specialists help organisations navigate the trade-offs between performance, security, and scalability, ensuring that data strategies align with long-term business needs.

For transactional operational applications, system developers can often manage data storage concerns sufficiently to meet quality expectations. However, for data-heavy projects—such as data warehouses, analytics, and large-scale insights-driven systems—engaging a database developer remains highly recommended. Their expertise ensures that data remains performant, secure, and well-governed, supporting both current and future requirements.

### Front End Developers

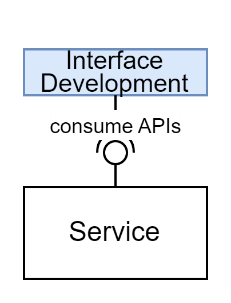


Figure 7: Front End / User Interface development

Front-End Developers specialise in building the **browser-based interface** of an application. Their focus is on **presentation, usability, and accessibility**, ensuring that users can effectively interact with the system. They work within the constraints of **the browser environment**, integrating with backend services but not defining business logic, data architecture, or security.

While they ensure applications function smoothly across devices, their expertise is typically limited to UI implementation, performance optimization, and user interaction rather than broader system design or backend logic. Their role is essential for user experience but does not extend to the structural or operational aspects of the system.

### System Developers

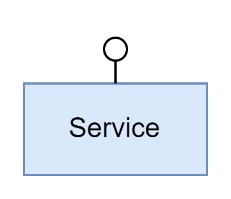


Figure 8: Service development

System developers are responsible for designing and building new systems from scratch and major application features. Their work is iterative, involving continuous refinement, breaking, and rebuilding components as part of the development lifecycle.

Unlike other developers who work on either the top visible layer of a platform (platform developers), in between systems (integration developers), under systems (deployment developers), or beside them (test developers), system developers focus on the system itself. They ensure that systems leverage infrastructure to support technical services first, in order to in turn build capabilities to deliver business and user functionality, all while addressing key quality concerns such as scalability, security, maintainability, and performance.

System development requires a deep understanding of software architecture principles, patterns, requirements and constraints, and the ability to anticipate both short-term delivery needs while defining the right locations and sufficient space for long-term system evolution.

Their work is foundational—without them, the other types of developers lack a system to deliver, test, maintain, integrate, or extend.

### Full-stack Developers

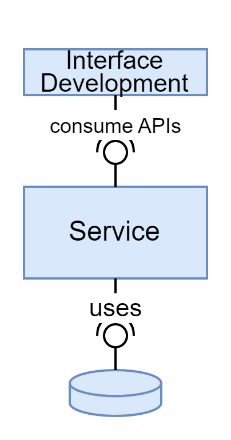


Figure 9: Full stack development

Developers who can build infrastructure, data storage, technical and business services, and front-end presentation layers are often referred to as full-stack developers. However, the term is frequently self-applied and does not always indicate full proficiency across all layers. Some "full-stack" developers are primarily front-end specialists with additional experience in platform development (e.g., extending systems like Salesforce or Drupal) rather than having deep expertise in system architecture, security, or backend infrastructure.

While a true full-stack developer can design and implement an entire system, in practice, most have a primary strength in either front-end or backend development while maintaining working knowledge of the other areas. Due to their broad technical understanding, full-stack developers often act as team leads, capable of orchestrating work across disciplines and bridging gaps between specialised developers.

## Inconsistencies of Terms

Partly due to the IT industry being younger than established fields like medicine or civil engineering, the terminology used remains inconsistent across organisations, with role definitions varying widely.

Hence, much like the overuse of the term "architect"—which is often misapplied to roles that function more like real estate agents, adept at finding tenancy in pre-existing solutions rather than having the skills to design systems from the ground up — terms like “scripter”, "coder," "developer," and "engineer" are frequently used in different context with unclear distinction.

That said, the term *engineer* is more commonly used to describe technicians who peripherally work around systems. They are commonly found in maintenance, DevOps, automation, and operations roles, focusing on efficiency, automation, resilience, and scalability. They may also develop a small amount of scripts to deploy systems, and configure others -- but their use is to contribute to ensuring systems remain maintainable, robust under load, deployments are seamless, and the underlying infrastructure is reliable, as opposed to developing new functionality.

The term *developer* on the other hand, tends to be used to describe those who develop new *functionality* – either an existing platform, or making a new service.

## Inconsistencies of Capability

Because software development remains a relatively young industry, career progression does not always lead to a broad understanding of system design. Some developers gain computer science training, but this often focuses on algorithms rather than system architecture. Others follow an on-the-job progression, moving from junior to intermediate, senior, and team lead roles. However, neither path guarantees an ability to design systems holistically.

At no stage is there certainty that a developer will mature beyond perceiving each system as an isolated project rather than recognising the broader patterns that underpin IT systems. Many developers, even at senior levels, approach software as though every problem is entirely novel, when for the most part, systems are approximately 80% systematic assemblies of well-known core services, with less than 10% of the system being unique to the business function, service, or algorithm it implements, with maybe another 10% being the art of marrying the two in an usable and accessible manner.

Essentially, the difference between an experienced developer and one who has matured into a broader systems thinker is their ability to distinguish between novelty and standardisation—knowing what patterns are available and well trodden, when to follow them to reduce the effort and risk of building novel work.

Broad system thinkers naturally progress from team leads to solution architects. Beyond defining the right balance of novel and standard work, and reducing their direct coding, they must communicate effectively with stakeholders while articulating the system’s design in a way that allows different trades to work independently. By providing clear blueprints, developers, engineers, and operations teams can focus on their respective domains with confidence that their work will integrate seamlessly.

Enterprise architects move beyond recognising technology patterns to understanding the broader patterns of how people collaborate and interact with technology portfolios. Ideally, they use this insight to guide strategic decision-making—though, if too detached from real systems design, they risk slipping into becoming real estate agents of vendor products and promises.

## Risks

While all development roles involve coding, only system developers have experience designing and developing full systems. Other roles are critical in maintaining, extending, integrating, or deploying software, but they are not trained in system architecture. Assigning a non-system developer to build a system from scratch increases the risk of overlooked architectural considerations, leading to suboptimal outcomes, technical debt, and costly rework. If the system lacks proper architectural foundations, system developers must compensate, taking on additional responsibilities across security, scalability, data integrity, and deployment—domains typically outside the expertise of other developers.

However, even when roles are correctly assigned, project success depends on the soundness of the task itself. A well-resourced team working on an ill-conceived architecture, misapplied platform, or rigid system will still struggle. Architects who assume that mature systems share common features often default to prebuilt platforms, assuming they reduce uncertainty compared to custom solutions, rather than evaluating whether they truly align with the organisation’s current needs and can evolve for an unknown future.

As an example, many customer relationship management (CRM) platforms are structured around sales pipelines and one-to-one customer interactions within a single organisation, making their core data model fundamentally unsuitable for managing relationships across multiple roles and organisations. Force-fitting an organisation’s needs into a system designed for a different purpose often results in significant compromises, leading to data model rigidity, costly workarounds, and functional misalignment. Similarly, some enterprise-grade data tools are designed for historical batch processing rather than real-time interactions, leading to brittle integrations when misapplied. These issues arise not from developer misalignment but from tasks being framed around the wrong technical choices.

Using platform developers to extend prebuilt systems may appear to provide short-term gains but often locks organisations into rigid structures that constrain long-term adaptability. However, poor platform choices frequently result in overly rigid data models that require expensive workarounds, misaligned workflows that force users to adapt to the software rather than the other way around, technical debt from excessive customisation leading to high maintenance costs, and limited interoperability with external systems due to platform-imposed constraints.

Thus, ensuring the right developers are assigned must go hand in hand with ensuring the task is defined correctly. A poor architectural decision will create inefficiencies, no matter how well resourced. Organisations must scrutinise not just who is doing the work, but whether the work itself is structured for success of project delivery and long-term operation inclusive of adaptability as required for an unpredictable future.

## Conclusion

Effective project resourcing requires a clear understanding of developer roles and their distinct responsibilities. Misaligning these roles—such as assigning a support or integration developer to system development—can lead to structural flaws, inefficiencies, and costly rework. Similarly, assuming system developers can handle deployment, testing, and integration without specialised support can create bottlenecks and slow delivery.

Beyond developer roles, organisations must also ensure architectural decisions align with long-term business needs. While leveraging existing platforms can accelerate development, they should be chosen for their fit with organisational objectives rather than as a default solution. Architectural oversight should balance flexibility, maintainability, and a wariness of vendor lock-in to prevent costly misalignment.

By correctly assigning developers to their strengths and ensuring architectural decisions are grounded in practical, adaptable system design, organisations can mitigate project risks, optimise resources, and build systems that support both immediate functionality and long-term evolution.

Appendices

Appendix A - Document Information

### Authors and Collaborators

Sky Sigal, Solution Architect

### Versions

* 1. Initial Draft
  2. Incorporation of Feedback

### Images

[Figure 1: Common developer categories 3](#_Toc190522524)

[Figure 2: Integration Development 4](#_Toc190522525)

[Figure 3: Platform Development 5](#_Toc190522526)

[Figure 4: Deployment Pipeline Development 5](#_Toc190522527)

[Figure 5: Test automation development 6](#_Toc190522528)

[Figure 6: Database Development 7](#_Toc190522529)

[Figure 7: Front End / User Interface development 8](#_Toc190522530)

[Figure 8: Service development 8](#_Toc190522531)

[Figure 9: Full stack development 9](#_Toc190522532)

### Tables

**No table of figures entries found.**

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
| Russell Campbell, Technical Manager |  |
| Amy Orr, Enterprise Architect |  |
| Rebecca Clark, Business Analyst |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Standards

ISO-25010

: …

ISO-25012

: …

ISO-25022

: …

### Acronyms

Refer to the project’s Glossary.

IT

: acronym for Information, using Technology to automate and facilitate its management.

##### ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.

### Terms