2026

WRANGU

A PATTERN LANGUAGE FOR SERVICENOW DELIVERY PROJECTS

(MVP Sampler)

TABLE OF CONTENTS

MVP SAMPLER EDITION

This book presents the Minimum Viable Pattern (MVP) set from "A Pattern Language for ServiceNow Delivery Projects" by Philip Swann, Head of Technical Consulting, Wrangu.

When you have finished reading this, you will be able to start applying this pattern language in your delivery projects and in doing so – improve your effectiveness. If you share it with your team the improvements should become exponential as the barriers in communication are broken down.

THANK YOU AND WELCOME

Thank you for downloading this MVP Sampler of A Pattern Language for ServiceNow Delivery Projects.

If you picked this up as part of **"Klepto-Coding: The Art of Stealing (and Other Secrets)",** treat it as your official permission slip. You are encouraged to read it, question it, reuse it, adapt it, and steal from it - intelligently.

This sampler is provided as a free conference handout and download. It is intentionally concise, practical, and opinionated. Its purpose is not to teach everything - but to show how this way of thinking works.

WHAT "MVP" MEANS HERE

In this context, MVP = Minimum Viable Pattern.

This sampler contains the minimum number of patterns required to:

- understand the philosophy of the language
- see how patterns relate to one another
- apply them immediately to real ServiceNow delivery work
- decide whether the full language is useful to you

Each pattern included here stands on its own - but more importantly, they begin to form a language when taken together.

WHAT YOU'RE LOOKING AT

This document is a curated extract from the full book **A Pattern Language for ServiceNow Delivery Projects** – by Philip Swann.

The complete language contains:

- significantly more patterns
- richer cross-linking between patterns
- deeper coverage across the full delivery lifecycle

The goal of the sampler is not completeness. The goal is demonstration.

If this resonates, the full language already exists to go much further.

WHAT IS A “PATTERN” IN THIS LANGUAGE?

A pattern is not:

- a rule
- a template
- a best practice checklist
- or a mandatory process step

A pattern describes:

- a recurring problem in ServiceNow delivery
- the forces that make it difficult
- a proven response that has worked repeatedly in real projects
- and the outcomes that tend to follow when it is applied consciously

Patterns capture experience - not theory.
They don’t tell you what you must do.
They help you recognise what situation you are in and what tends to work there.

As a result:

- patterns can be combined
- adapted
- partially applied
- or deliberately ignored (with eyes open)

This is a language for thinking and communicating about delivery, not enforcing dogma.

Why a Pattern Language?

ServiceNow delivery fails less often due to lack of technical skill - and more often due to:

- unclear scope
- misaligned expectations
- unexamined assumptions
- accidental communication structures
- undocumented decisions
- and invisible trade-offs

A pattern language makes these visible, discussable, and repeatable - without pretending every project is the same.

HOW TO USE THIS SAMPLER

- Read it non-linearly.
- Jump to the problems that feel familiar.
- Notice how patterns reference one another.
- Try using just one pattern deliberately on your next project.

If a pattern gives you language you didn't have before - it's doing its job.

FOREWORD

By Chuck Tomasi

WHY BEST PRACTICES ARE IMPORTANT

For me, technical best practices have two purposes.

First, they can make development faster by offering a way to avoid issues based on lessons learned from those who came before us. Consider trying to write a song when all you know is how to play notes. You will likely be able to string together a series of notes that technically constitutes a song, but it wouldn't have the same impact or elicit the same emption if you had studied other composers/artists to learn from their experience what works and what doesn't. Don't just jump in and start coding until you understand the outcomes of what you're building and the best approach to do so. Don't do things the hard way, use technical best practices to avoid mistakes.

The second purpose of technical best practices is that they offer a way to improve maintainability through a set of standards. This may be coding styles (where to place curly braces, newline spacing between code blocks, or commenting styles) or it could be as subtle as naming tables and fields - technical best practices are a great way to define standards so developers can work together to maintain code. After 40 years of coding, I can honestly say one of the biggest hurdles to get over is trying to "unwind" someone else's work - be it code or architecture. First, you need to understand before you can make changes. By using the same set of best practices, you make it easier to understand.

"Those who cannot remember the past are condemned to repeat it" - George Santayana

D5 MODEL

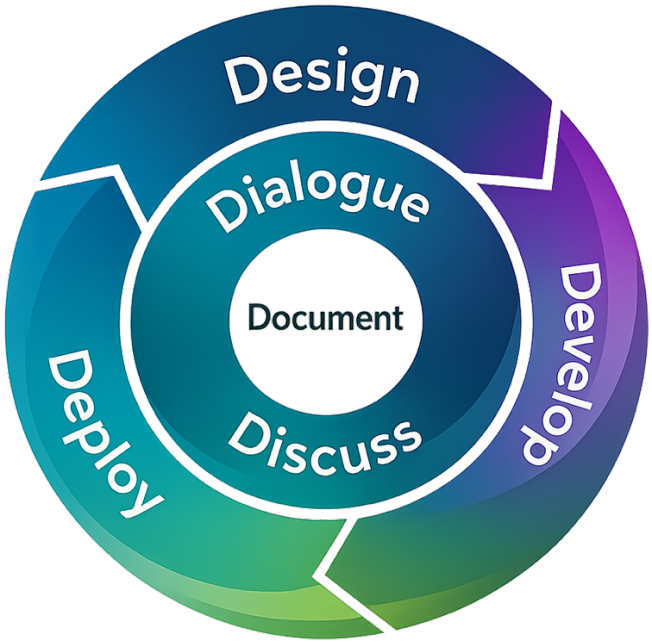
DEFINITION

The D5 Model defines the order of the pattern language used throughout ServiceNow delivery projects.
It is not a phase model or methodology. It describes how understanding, intent, and delivery coherence are established, maintained, and evolved over time.

The language is ordered across five domains, described as a living system:

- Document provides stable grounding.
- Dialogue keeps understanding current.
- Design, Develop, and Deploy spin continuously as delivery evolves.

This structure allows projects to adapt over time without losing coherence, intent, or quality, supporting long-running engagements, team rotation, and complex operating environments.



DOCUMENT – CENTRAL GROUNDING

Document sits at the centre of the language as the persistent grounding for delivery.

Document is not just the starting point. It remains active throughout the entire lifecycle, providing:

- Context
- Shared understanding
- Recorded intent
- Traceability and governance

As delivery progresses, Document is continually referenced, validated, and updated to reflect evolving understanding, decisions, and outcomes. It anchors the project regardless of phase, personnel, or environment.

DIALOGUE (DISCUSS) – CONTINUOUS INTERFACE

Dialogue is the expansive interface between people and documentation.

Dialogue runs continuously through all stages of delivery, enabling:

- Understanding to surface before commitment
- Assumptions to be challenged
- Context to evolve safely without fragmentation

Dialogue ensures that Document remains alive rather than static, and that Design, Development, and Deployment remain grounded in shared meaning rather than assumption.

DESIGN, DEVELOP, DEPLOY – ITERATING WHEEL

Surrounding the central grounding of Document, the outer domains form a continuously iterating cycle:

- Design shapes intent based on documented context and live dialogue.
- Develop realises design intent without eroding meaning or quality.
- Deploy introduces change safely and predictably into live environments.

These domains do not operate independently:

- You cannot deploy without developing.
- You should not develop without design.

Design cannot be done properly without understanding what already exists and has previously been deployed.

Each deployment becomes new context that feeds back into Document and Dialogue.

As a result, delivery progresses through repeated cycles of Design → Develop → Deploy, each iteration grounded by Document and informed through Dialogue.

DOCUMENT

This domain contains patterns that make delivery **legible**: the artefacts, boundaries, and recorded decisions that stop progress relying on memory, proximity, or personal authority. Use these patterns to create stable reference points that can survive time, handovers, pressure, and change.

SCOPE

Intent

Define and maintain a shared understanding of the boundaries within which delivery operates.

Scope clarifies what work is included, excluded, and conditionally permitted, translating commercial authority into delivery-level understanding without attempting to over-specify behaviour or solutions.

Scope exists to answer the question:

- *"What are we actually meant to do - and under what conditions?"*

When to Use

Use this pattern:

- At the beginning of delivery work
- Before design commitments are implied
- When interpreting commercial artefacts into delivery reality
- When new ideas, assumptions, or requests emerge
- When stakeholders appear to be working to different expectations
- Whenever delivery friction or misalignment appears
- Scope is not a one-time activity.

Forces

- Broad commercial language vs concrete delivery decisions
- Momentum vs clarity
- Discovery vs commitment
- Evolving understanding vs fixed expectation
- Multiple stakeholder interpretations
- Helpfulness vs scope erosion

Left unmanaged, these forces result in drift, tension, and loss of confidence.

How it Works

Express scope as **explicit boundaries**, not exhaustive detail

Clearly distinguish between:

- What is included
- What is excluded
- What is conditional or subject to further agreement
- Use scope as a reference point, not a weapon
- Revisit and restate scope as understanding evolves
- Surface scope tension early and visibly, rather than absorbing it silently

Scope does not block progress - it protects intent and focus.

Outcome

When the Scope pattern is applied effectively:

- Stakeholders share a common understanding of project boundaries
- Delivery teams can make confident day-to-day decisions
- Scope creep is detected early, not justified later
- Commercial and delivery conversations remain aligned
- Trust improves through transparency and predictability
- Projects focus effort on value, not unplanned obligation

Most importantly, delivery stops relying on assumption.

Examples

A stakeholder requests an additional workflow during a workshop.

Instead of debating feasibility, the team references Scope to determine whether this is:

- in scope
- out of scope
- conditionally permitted subject to change control

A “quick fix” is requested late in the build phase.

Scope is used to decide whether this fits within agreed boundaries or requires formal approval.

Related Patterns

- Commercial Cover
- Statement of Work
- Project Charter
- RIDAC
- Channels of Communication
- Project Status Report

COMMERCIAL COVER

Intent

Ensure that all work undertaken has explicit authority, whether that work is billable, complimentary, exploratory, advisory, delivery-based, or undertaken at risk. Commercial Cover establishes a transparent basis for why work is being done, under what arrangement, and with whose authorisation, protecting both delivery teams and client relationships.

Commercial Cover is not a single document. It is a pattern that describes authority to act.

A Statement of Work is one manifestation of this authority, but not the only one.

When to Use

Use this pattern whenever work is expected, requested, or already underway, including:

- At the start of delivery engagements
- When scope, expectations, or ways of working begin to shift
- When new activities are requested that were not previously discussed
- When delivery teams feel implicit pressure to “just help”
- When transitioning between types of engagement or working models

Commercial Cover is relevant whether or not a formal SOW exists.

Forces

- Authority vs assumption
- Client urgency vs contractual clarity
- Helpfulness vs liability
- Sales intent vs delivery reality
- Flexibility vs governance
- Long-term trust vs short-term accommodation

Without clear commercial authority, delivery decisions become implicit, personal, and difficult to defend.

How it Works

- Establish and make visible the basis on which work is authorised
- Ensure delivery teams understand what they are permitted to do, and just as importantly, what they are not
- Use the recognised commercial artefact(s) for the engagement as reference points, without treating them as instructional truth
- Surface tension early when requests fall outside the current authority to act
- Treat changes to authority as explicit commercial decisions, not delivery side-effects
- Commercial Cover does not prevent change - it ensures change is conscious.

Outcomes

When Commercial Cover is applied effectively:

- Delivery teams operate with confidence rather than inference
- Unpaid or misaligned work is reduced
- Commercial and delivery conversations remain aligned over time
- Governance and escalation rely on evidence, not recollection
- Trust improves through transparency rather than enforcement
- Most importantly, delivery avoids relying on goodwill alone.

Examples

A consultant is asked for a “quick fix” on Teams by a client stakeholder outside the project scope. A 30-minute task spirals into hours. The project later faces

commercial disputes because the stakeholder assumed it was included. Once the Account Manager reviewed the engagement terms, a Change Order was required, but the work had already been delivered informally - resulting in write-offs and strained trust.

Related Patterns

- Scope
- Statement of Work
- Project Charter
- RIDAC
- Channels of Communication
- Project Status Report

DIALOGUE

This domain contains patterns that keep understanding **current**: how people align, surface assumptions, and turn conversation into reliable shared meaning. Use these patterns to prevent 'agreement by fatigue' and ensure Document stays alive while delivery evolves.

INTRODUCTIONS

Intent

Establish shared human, organizational, and contextual understanding whenever people come together to work - before applying that understanding to a specific endeavor.

Introductions are a general interaction pattern, not one invented for delivery projects. They exist wherever people collaborate: meetings, workshops, programs, incidents, communities, and teams.

At their core, Introductions enable people to understand who is present, how they see themselves, and how they expect to contribute.

Within a ServiceNow delivery project, this pattern is applied deliberately to ground the work in human context. It ensures that roles, perspectives, authority, experience, and expectations are made visible early - so project decisions are shaped by understanding rather than assumption.

Introductions do not exist to satisfy etiquette. They exist to create orientation.

When to Use

Introductions should be used:

- Whenever people who may not share full context come together
- At the beginning of workshops, meetings, or collaborative sessions
- When new individuals join an existing group
- When the purpose, authority, or expectations in the room are unclear

Within delivery projects, this typically includes:

- Project kick-offs (internal and external)
- The start of workshops or discovery sessions
- Mid-project onboarding of new stakeholders
- Governance or escalation meetings where new participants are present

Introductions may be repeated multiple times across a long-running engagement as context and participation evolve.

Forces

Several tensions shape the need for effective Introductions:

- Social Convention vs Useful Context
- Polite introductions often lack information that is practically relevant.
- Title vs Reality
- Formal roles do not reliably indicate influence, authority, or ownership.
- Speed vs Orientation
- Teams are often eager to begin “the real work” before understanding each other.
- Familiarity Bias
- Some participants know each other well; others are outsiders.
- Human Context vs Task Focus
- Delivery discussions often jump straight to artefacts, skipping people.

These forces exist in all collaborative settings. In delivery projects, they directly affect scope, decisions, governance, and trust.

How it Works

Treat Introductions as Orientation, Not Formality

The facilitator frames Introductions as a way to help people work well together - not as ceremony or biography.

Invite Relevant Self-Description

Each participant introduces themselves in a way that answers questions the group will otherwise infer badly, such as:

- Who they are and what they are responsible for
- How they relate to the subject or system being discussed
- Where their authority, input, or constraints lie
- What they care most about in the context of this collaboration

The structure may be lightweight, but the intent is clarity.

Take Notes and Observe

Introductions are active listening moments.

Within delivery, this means:

- Capturing roles, influence, expectations, and concerns
- Noting mismatches between stated role and apparent authority
- Observing dynamics: who defers, who leads, who is cautious

This information becomes contextual input, not just background.

Apply the Context to the Work

The information gathered during Introductions is then applied to the activity at hand:

- Shaping facilitation and discussion
- Informing governance and escalation paths
- Guiding communication choices
- Adjusting pacing, depth, or framing of decisions

Introductions do not end when people stop speaking - they continue through application.

Outcome

When Introductions are applied effectively:

- People understand who is in the room and why
- Assumptions about authority and responsibility are reduced
- Collaboration becomes more deliberate and inclusive
- Facilitation improves because context is visible
- Decisions are made with awareness of human and organizational reality

Within delivery projects, this results in fewer misunderstandings, clearer governance, and stronger early alignment - but these are consequences, not the primary purpose.

Examples

- A cross-functional meeting reveals that several attendees are advisors, not decision-makers, preventing false consensus.
- Introductions expose that a “non-technical” stakeholder owns final approval for integrations, shaping how design discussions are handled.
- A newly introduced compliance lead surfaces regulatory pressure that reframes risk tolerance for upcoming deployments.

Related Patterns

- Take Note - captures contextual information revealed through Introductions
- Channels of Communication - shaped by stakeholder preferences and roles
- Seating Arrangement – Strategic Placement - informed by authority and influence
- Story Listening - human context sharpens what is heard and prioritized
- Shape the Narrative - people and perspectives anchor the evolving story

CHANNELS OF COMMUNICATION

Intent

Ensure that communication choices on a delivery engagement are intentional, appropriate, and governed, without forcing a single channel or tool.

This pattern exists to prevent fragmented understanding, silent sources of truth, and misaligned expectations by making communication explicit rather than accidental.

Channels of Communication is not about mandating tools.

It is about validating which channels are appropriate, for what purpose, and for whom.

When to Use

Apply this pattern:

- At the start of an engagement, when communication norms are first set
- When new stakeholders enter the delivery space
- When information begins to fragment across chat, email, meetings, and documents
- When delivery rhythm or pressure changes
- When confusion arises about where to share, find, or trust information

Communication channels are expected to evolve over time.

Forces

- Speed vs durability
- Transparency vs noise
- Convenience vs governance
- Momentary interaction vs enduring record
- Tool capability vs human habit

Left unmanaged, these forces create parallel realities, lost decisions, and retrospective disagreement.

How it Works

Explicitly validate which channels are approved for use on the engagement

For each channel, make clear:

- Its purpose
- Its intended audience
- The type of information it is appropriate for
- Allow multiple channels to coexist where they are fitness-for-purpose
- Treat informal channels as distribution mechanisms, not authoritative records
- Ensure that anything affecting scope, risk, decisions, commitments, or direction is captured in an authoritative artefact

No communication channel is authoritative by default.

Outcomes

When Channels of Communication is applied effectively:

- Stakeholders know where to share information and where to look
- Delivery teams stop chasing individuals across tools
- Informal updates no longer replace formal records
- Decisions remain visible even as conversations move
- Communication adapts without fragmenting delivery memory

Most importantly, the project avoids creating truth by proximity or convenience.

Examples

A delivery team uses a group chat for quick coordination during build.

Key decisions and scope clarifications are referenced back to authoritative artefacts rather than being treated as “agreed because it was said”.

A stakeholder sends an important request via direct message.

The request is acknowledged, then surfaced formally so it can be assessed, recorded, and acted on explicitly.

Related Patterns

- Project Repository

- Project Status Report
- RIDAC
- Scope
- Commercial Cover
- Take Note
- Shape the Narrative

PEER REVIEW

Intent

Ensure that delivery outcomes are correct, coherent, defensible, and valuable by introducing a deliberate second-pair-of-eyes review before work is promoted, accepted, or relied upon.

Peer Review exists to preserve intent, reinforce quality, and reduce delivery risk without relying on individual judgement or memory.

Peer Review is not punitive, not clerical, and not personal. It is a shared quality practice rooted in collaboration, evidence, and accountability.

When to Use

Use this pattern whenever:

- Work will be promoted, deployed, or treated as complete
- Changes materially affect behaviour, data, architecture, integrations, or controls
- Design intent must remain understandable beyond the original contributor
- Delivery risk, uncertainty, or complexity increases
- Teams include mixed experience levels
- Confidence, not just correctness, matters

Peer Review is continuous and risk-scaled. It may be lightweight or deep depending on context, but it is never optional.

Forces

Peer Review resolves several competing forces:

- Speed vs quality

- Individual judgement vs shared standards
- Learning vs throughput
- Delivery momentum vs long-term maintainability
- Memory vs evidence

Left unmanaged, these forces lead to fragile implementations, inconsistent standards, hidden risk, and rework justified after the fact.

How it Works

Peer Review introduces a deliberate pause in delivery flow-long enough to reason, short enough to maintain momentum.

Effective Peer Review:

- Makes implementation, intent, and impact visible
- Validates that outcomes align with expectations and accepted standards
- Surfaces trade-offs, assumptions, and risks explicitly
- Preserves reasoning so decisions remain explainable over time
- Scales depth of review based on risk and complexity
- Reinforces shared ownership of quality rather than individual heroics

Peer Review commonly applies the SIMPLE framework as its primary review lens.

SIMPLE does not define standards; it ensures that applicable standards-project, organisational, ServiceNow, and Wrangu baseline-are explicitly considered.

Outcome

When Peer Review is applied effectively:

- Quality is reinforced proactively rather than inspected retrospectively
- Design intent survives team rotation and project duration
- Risks surface earlier, when they are cheaper to address
- Delivery decisions are defensible and explainable over time
- Junior consultants learn safely through visibility and feedback
- Teams rely on evidence instead of recollection

Most importantly, delivery stops relying on implicit trust alone.

Examples

- A completed story is reviewed to confirm behaviour, intent, and acceptance before promotion
- A complex design choice is reviewed early to prevent downstream rework
- A low-risk change receives a lightweight review, while higher-risk work is assessed in greater depth
- A reviewer identifies an undocumented assumption that materially affects deployment or operations

Related Patterns

- Configuration vs Customisation
- Project Master Batch
- Deployment Plan
- RIDAC
- Project Status Report
- Shape the Narrative

DESIGN

This domain contains patterns for shaping intent into a coherent solution direction before you commit to build. Use these patterns to stabilise understanding, explore options deliberately, and make trade-offs visible - so 'design' remains explainable long after the workshop ends.

CHART A MAP

Intent

Create a shared, validated understanding of how work actually happens when existing documentation is missing, unclear, outdated, or contradicted by lived reality.

Chart a Map exists to stabilise understanding early enough to proceed with confidence, without pretending certainty or completeness.

It enables teams to move forward grounded in evidence rather than assumption.

When to Use

Use this pattern when:

- There is little or no usable process documentation
- Existing documentation is incomplete, contradictory, or no longer trusted
- Stakeholders describe the same process differently
- Client walkthroughs reveal "how it really works" rather than how it is documented
- Delivery needs to progress before full formal documentation exists
- Early understanding is required to inform scope, stories, or design decisions

Chart a Map is particularly valuable at moments where discovery and commitment begin to overlap.

Forces

Several tensions make this pattern necessary:

- Formal policy vs lived practice

- Speed to progress vs confidence in understanding
- Verbal explanation vs shared reasoning
- Partial knowledge vs delivery commitment
- Distributed perspectives vs a single narrative

Left unmanaged, these forces produce fragile designs, scope disputes, and rework justified after the fact.

How it Works

Chart a Map captures and stabilises understanding through deliberate sense making:

- Capture reality as described and observed, not as it “should” be
- Translate conversations, walkthroughs, and examples into simple visual or structured representations
- Focus on flow, ownership, decisions, exceptions, and dependencies
- Make assumptions, gaps, and contradictions visible rather than smoothing them away
- Validate the map collaboratively by asking “Is this what actually happens?”
- Iterate the map as understanding improves, without striving for premature perfection

The goal is not modelling precision, but reasoning clarity.

Outcome

When Chart a Map is applied effectively:

- The team gains a shared, grounded understanding of current behaviour
- Hidden complexity, dependencies, and exceptions surface early
- Assumptions become explicit and discussable
- Downstream work starts from evidence rather than inference
- Delivery progresses with alignment rather than guesswork

Most importantly, the project stops relying on undocumented tribal knowledge.

Examples

- A stakeholder describes an approval process as “simple”; mapping exposes multiple escalation paths and role variations
- Regional teams provide conflicting explanations of the same workflow; the map makes differences visible and resolvable
- A perceived mature process is walked through and found to lack agreed sequencing, ownership, or controls

Related Patterns

- Take Note
- Draw It Out
- Make It Make Sense
- Use Case Model
- Baseline Starter Stories
- Scope
- Project Repository

4+1 DOMAIN MODEL

Intent

Provide a single, shared way of reasoning about a ServiceNow solution that keeps understanding coherent across roles, decisions, and time.

The 4+1 Domain Model exists to ensure everyone involved can answer:

- *“How does this solution work, and why is it designed this way?”*

It avoids fragmented understanding by holding multiple necessary perspectives together without forcing them into a single document, diagram, or abstraction.

When to Use

Use this pattern whenever:

- Multiple stakeholders view the solution from different perspectives
- Design discussions begin to fragment or feel circular
- Decisions are being made without shared context
- New team members need to understand the solution beyond isolated artefacts

- Delivery spans time, iterations, or team change

The model is relevant from early discovery through to live operation, not just during "design".

Forces

- Multiple perspectives vs shared understanding
- Early clarity vs premature commitment
- Technical depth vs accessibility
- Evolving delivery vs stable reasoning
- Conversation richness vs loss of rationale

Left unmanaged, these forces cause design drift, rework, and decisions that cannot later be explained.

How it Works

The model separates how we reason about the solution into five connected views, each answering a different question, while remaining explicitly linked.

The "+1" - Use Cases

At the centre sits the Use Case view:

- who the solution serves
- what they are trying to achieve
- how they interact with it

This is the anchor for meaning. Every other view derives relevance from it.

Logical View

What exists in the solution: key entities, responsibilities, and boundaries.

This view supports reasoning about structure, ownership, and correctness.

Process View

How work flows through the system: states, transitions, decisions, and exceptions.

This view reflects how work actually happens, not how it is ideally described.

Development View

How intent is realised on the platform: configuration, extensions, and integration points.

This connects reasoning to implementation and underpins review and governance.

Environment / Physical View

Where the solution lives and moves: environments, deployment paths, and operational constraints.

This grounds design in delivery reality and risk.

Keeping the Model Alive

The model is:

- referenced during discussions and decision-making
- updated when material understanding changes
- stored and shared as an evolving reference, not a final artefact

The goal is not diagram perfection, but visible, coherent reasoning.

Outcomes

When the pattern is applied effectively:

- Stakeholders share a common understanding of the solution
- Design discussions become clearer and more productive
- Decisions remain explainable over time
- New team members onboard faster
- Design intent survives beyond individuals

Most importantly, the solution stops relying on memory or individual interpretation.

Examples

- During workshops, multiple stakeholders describe the same process differently.
- Use cases and process views make the differences explicit and resolvable.
- A development decision is challenged late in the project.
- The logical and development views show why the decision was made and what trade-offs were accepted.
- Deployment risk is raised close to go-live.
- The environment view links design assumptions directly to deployment reality.

Related Patterns

- Use Case Model
- Chart a Map
- Solution Overview (Project-Based)
- Scope
- Configuration vs Customisation

RIGHT-CLICK EVERYTHING

Intent

Enable rapid understanding, confident decision-making, and high-quality design by examining what already exists in the ServiceNow platform before attempting to create anything new.

This pattern encourages practitioners to treat the platform itself as the primary learning surface. By right-clicking, inspecting, and exploring existing records, configurations, scripts, flows, and artefacts, consultants ground their work in proven paradigms rather than assumption or invention.

Right-Click Everything exists to prevent reinventing the wheel and to promote design that is native, consistent, and evolvable.

When to Use

Use this pattern:

- Whenever you encounter a new problem, requirement, or behaviour you do not yet fully understand
- Before designing or building custom solutions
- When working in unfamiliar product areas, modules, or tables
- During troubleshooting or forensic analysis
- When inheriting or reviewing someone else's implementation
- When assessing whether a requirement is already solved, partially solved, or nearly solved by the platform

If the question is "How should this work?", your first response should be: "What already works this way?"

Forces

Several forces make this pattern essential:

- Platform richness - ServiceNow already contains mature solutions to many problems
- Time pressure - invention feels faster than investigation (but usually isn't)
- Knowledge asymmetry - newer consultants may not know what already exists
- Consistency vs creativity - originality is often valued over alignment
- Upgrade safety - custom patterns increase long-term maintenance risk
- Confidence - understanding existing implementations reduces uncertainty

These forces pull teams toward unnecessary customisation unless actively resisted.

How it Works

Assume Nothing Is Original

Start from the position that the platform already contains something conceptually similar to the problem you are facing.

Right-Click and Inspect

Use right-click actions to:

- View table structures and dictionary entries
- Inspect business rules, client scripts, script includes
- Examine workflows, flows, sub-flows, and actions
- Review UI policies, UI actions, notifications, and SLAs
- Explore related lists, references, and M2Ms

Follow the Thread

Navigate outward:

- From records to scripts
- From scripts to utilities
- From flows to underlying tables

- From configuration to data and back again

Identify the Paradigm

Look for:

- Naming conventions
- Structural patterns
- Repeated logic
- Design intent, not just implementation detail

Steal Intelligently

Apply the paradigm - not the artefact - to your new context.

Adapt patterns, do not copy blindly.

Design From What Is Known

New solutions should feel like they already belong in the platform.

Outcome

- Designs align naturally with ServiceNow's native behaviours
- Customisation is reduced without sacrificing capability
- Solutions are easier to explain, maintain, and upgrade
- Consultants gain confidence through understanding, not guesswork
- Knowledge transfer accelerates organically
- The platform becomes a living reference, not a black box

Examples

- A consultant unsure how to implement approval logic explores OOTB approval flows and adapts the same structure for a bespoke process.
- A developer diagnosing unexpected state changes inspects existing business rules on the same table to uncover implicit platform behaviour.
- A team designing notifications examines OOTB templates, categories, and layouts before defining their own.
- During workshops, questions about "best practice" are answered by showing how the platform already does it.

Related Patterns

- Configuration vs Customisation - supports informed decisions about deviation from OOTB.
- Make It Make Sense - applied after analysing multiple existing paradigms.
- Draw It Out - helps capture and communicate discovered patterns visually.
- Sanitize - critical when reusing ideas or structures from other contexts.
- Peer Review - validates that adopted paradigms are being applied correctly.

DEVELOP

This domain contains patterns for turning design intent into working ServiceNow behaviour without eroding quality, clarity, or upgrade safety. Use these patterns to keep build decisions consistent, reviewable, and defensible - especially under time pressure.

CONFIGURATION VS CUSTOMISATION

Intent

Provide a clear, defensible way to reason about deviations from out-of-the-box ServiceNow behaviour without relying on simplistic or misleading classifications.

This pattern exists to shift teams away from debating labels and toward making explicit, contextual, and reviewable design decisions that preserve quality, upgradeability, and trust.

When to Use

Use this pattern whenever:

- Delivery work deviates from default platform behaviour
- A design decision raises concerns about long-term ownership, upgradeability, or governance
- Stakeholders ask “is this configuration or customisation?”

- Technical teams feel pressure to justify decisions using language rather than reasoning
- Peer Review, escalation, or audit requires visibility of design trade-offs

This pattern is especially important under time pressure, where poor classification decisions are most likely.

Forces

- Apparent simplicity vs hidden impact
- Speed of delivery vs long-term operability
- Platform capability vs organisational constraints
- Governance expectations vs pragmatic delivery
- Short-term value vs future cost

Left unmanaged, these forces drive dogma, hidden risk, and decisions justified after the fact.

How it Works

Reject the Binary

Do not treat “configuration” and “customisation” as decision criteria.

They are descriptive terms, not measures of risk or value. Arguing about labels does not reduce impact.

Evaluate the Deviation Explicitly

Each meaningful deviation is assessed across a small, repeatable set of concerns, such as:

- Upgrade and maintainability
- Operational ownership and support model
- Reversibility
- Blast radius and coupling
- Alignment with the customer’s governance and operating model
- Alternatives considered, including native platform approaches

This replaces philosophical debate with structured reasoning.

Make the Trade-off Visible

Every significant deviation must be explicitly:

- Described (what is changing)
- Justified (why this approach was chosen)
- Risk-assessed (what could be affected)
- Mitigated (how risk is controlled)

If a deviation matters enough to debate, it matters enough to record.

Record the Decision

The reasoning and outcome are captured in the appropriate project artefacts and referenced during Peer Review.

Undocumented deviation is itself a delivery risk.

Validate Through Peer Review

Peer Review tests whether:

- The trade-offs are understood
- The risks are acceptable in context
- Alternatives were genuinely explored
- The decision is coherent and defensible

This removes reliance on individual judgement or authority.

Revisit Over Time

Acceptability is contextual, not permanent.

Decisions are revisited as scope, ownership, maturity, or governance expectations evolve.

Outcome

When this pattern is applied effectively:

- Teams stop arguing about labels
- Design decisions become explicit and defensible
- Hidden or accidental customisation is reduced
- Governance discussions improve in quality
- Peer Review focuses on impact rather than ideology
- Delivery choices align with real operating constraints

Most importantly, design decisions stop relying on assumption or memory.

Examples

An out-of-the-box workflow is retained deliberately after assessing ownership and future roadmap impact

A scripted extension is implemented with known blast radius and reviewed mitigation

A deviation is accepted early in delivery but revisited later as platform maturity increases

Related Patterns

- Peer Review
- RIDAC
- Right-Click Everything
- Solution Overview
- Project Status Report

GULASH

Intent

Make datatype intent visible, consistent, and trustworthy at the point of use, so that code can be read, reasoned about, reviewed, and safely evolved by people who did not originally write it.

GuLash exists to reduce ambiguity in JavaScript running on the ServiceNow platform by ensuring that variable names communicate what a value is, not just what it represents. This improves confidence during development, peer review, troubleshooting, and long-term maintenance.

The pattern optimises for readers over writers.

When to Use

Use this pattern when:

- Writing JavaScript where datatypes are not explicit (e.g. ServiceNow ES5)

- Working with Glide APIs and platform objects whose behaviour varies by type
- Code will be read or maintained by others, now or later
- Performing or preparing for Peer Review
- Diagnosing defects caused by incorrect assumptions about values
- Working in long-running engagements or rotating teams

GuLash is particularly valuable where mistakes are not syntactic, but semantic.

Forces

- Loose typing vs confidence in behaviour
- Speed of writing vs safety of reading
- Local convenience vs shared understanding
- Implicit knowledge vs visible intent
- Individual familiarity vs team comprehension

Without visible intent, teams rely on assumption, memory, or runtime discovery - all of which increase risk.

How it Works

Variable names explicitly encode the expected datatype of the value they hold.

Naming is applied consistently across the codebase, not selectively.
The prefix communicates how the value should be interacted with (e.g. which properties or methods are safe)
The remainder of the name expresses the semantic meaning, not the structure.
The goal is not mechanical compliance, but clarity under review and change.

GuLash does not replace understanding the platform - it makes understanding visible.

A Proven Vocabulary (Wrangu Default)

The following prefixes represent a proven, readable vocabulary used consistently across delivery projects. This is an instantiation of the pattern, not the pattern itself.

Datatype prefixes

- `arr` - Array
- `is` / `has` - Boolean
- `api` - Script Include class instance
- `ga` - GlideAggregate
- `gd` - GlideDate
- `gdt` - GlideDateTime
- `ge` - GlideElement
- `gr` - GlideRecord
- `obj` - Object
- `qc` - Query Condition
- `str` - String
- `id` - Sys ID
- `num` - Number
- `int` - Integer

Function and method prefixes

Functions begin with a verb and indicate their behaviour or return expectation, for example:

- get
- set
- check
- run
- process
- has / is

The exact vocabulary is less important than consistency and shared meaning.

Outcome

When GuLash is applied effectively:

- Reviewers can reason about behaviour without inspecting implementation
- Incorrect assumptions about values surface earlier
- Code becomes safer to extend, refactor, and debug
- Peer Reviews focus on intent and impact rather than guesswork
- Knowledge transfer improves without reliance on oral explanation

Most importantly, code stops relying on the reader "just knowing".

Examples

- A variable named incident requires inspection to determine whether it is a record, an identifier, or a string.
- A variable named `grIncident` communicates how it should be used immediately.
- A function that accepts `strUserId` and `grUser` makes misuse obvious during review, not at runtime.
- During Peer Review, reviewers can validate correctness by reading names, not tracing execution.

Related Patterns

- Peer Review
- SIMPLE
- Right-Click Everything
- Consistent Variable Datatypes
- Default Variable Definitions
- Guard Clause

DEPLOY

This domain contains patterns for introducing change safely into real environments: sequencing, readiness, communication, and recovery. Use these patterns to make releases predictable and explainable - and to reduce the gap between 'it worked in dev' and production reality.

DEPLOYMENT PLAN

Intent

Ensure that changes are introduced into target environments in a controlled, predictable, and explainable way, with sequencing, dependencies, readiness, and recovery made explicit.

The Deployment Plan exists to reduce deployment risk by turning change promotion from an act of trust into an act of preparation.

It creates shared confidence by making deployment intent visible before execution, rather than relying on memory, experience, or optimism at the point of release.

When to Use

Use this pattern whenever:

- Changes are promoted between environments
- Delivery confidence is fragile or contested
- Deployments include multiple components, dependencies, or contributors
- Manual steps, data movement, or sequencing matter
- Evidence of readiness, intent, or recoverability may later be required

This pattern is relevant regardless of deployment mechanism, customer operating model, or toolchain maturity.

Forces

The Deployment Plan resolves several competing forces:

- Speed vs safety
- Parallel development vs ordered release
- Automation vs unavoidable manual reality
- Memory vs evidence
- Local team confidence vs stakeholder trust

When left unmanaged, these forces lead to fragile releases, reactive recovery, and retrospective disagreement about what happened and why.

How it Works

The Deployment Plan establishes a single, coherent view of deployment intent before any change is applied:

- Make the scope of the deployment explicit: what is being introduced and where
- Make ordering visible: which changes depend on others and why sequencing matters
- Surface prerequisites and readiness conditions rather than assuming them

- Treat all deployment activity as first-class, including configuration, data, and manual steps
- Make timing awareness visible without false precision
- Define recovery expectations in advance, not during failure
- Capture context and rationale so the deployment can be understood after the fact

The plan does not prescribe tools or mechanics. It makes the logic of deployment inspectable.

Outcome

When the Deployment Plan pattern is applied effectively:

- Deployments become intentional rather than reactive
- Dependencies surface before they cause failure
- Manual steps stop being tribal knowledge
- Stakeholders gain confidence through visibility, not reassurance
- Reviews, governance, and escalations rely on evidence rather than recollection
- Teams retain a clear deployment narrative over time

Most importantly, deployment stops being an act of faith.

Examples

- A complex release with multiple update sets and data changes is sequenced explicitly, avoiding partial deployment and rollback confusion
- A customer challenges deployment risk late in the process; the plan provides a clear, defensible explanation of readiness and recovery
- A production issue occurs post-release; the plan allows rapid reconstruction of what changed and in what order

Related Patterns

- Environment Plan
- Deployment Batching
- Project Master Batch
- Peer Review
- Project Status Report

- RIDAC

CONCLUSION

WHAT HAPPENS NEXT

This sampler is not intended to be read once and set aside.
It exists to change what you notice - and therefore what you tolerate - when you are delivering ServiceNow work.
If it has worked, you should now find it harder to:

- accept unclear scope as “just how things are”
- let decisions remain implicit
- compensate silently for missing structure
- or hide uncertainty behind familiar terminology

You do not need organisational permission to begin using this language.
You only need to recognise a pattern you have already lived through.

Start with one that felt uncomfortably familiar.
Use it deliberately and name it out loud.
Pay attention to what becomes easier to discuss - and what stops being personal.
That is how delivery improves without relying on heroics.

This sampler is complete when it changes a conversation.
Everything beyond that is optional.

ABOUT THE AUTHOR

Philip Swann is Head of Technical Consulting at Wrangu, a ServiceNow partner specialising in complex, risk-sensitive delivery environments.

He has spent many years delivering, recovering, and governing ServiceNow engagements where success was constrained by real-world forces: commercial pressure, organisational ambiguity, regulatory risk, and evolving operating models.

Across those environments, the same problems appeared repeatedly - regardless of team, customer, or industry:
unclear scope, undocumented decisions, accidental communication structures, and good people compensating for weak foundations.

This pattern language emerged from recognising those repetitions and choosing to name them.

It reflects how delivery actually unfolds over time - not how it is often described in frameworks, methodologies, or tooling guidance.

Philip's work focuses on improving outcomes by making intent, structure, and trade-offs visible - so teams can reason about delivery deliberately, rather than relying on memory, authority, or optimism.

A Note on Authorship and Assistance

The ideas, patterns, and judgments in this language are human in origin. They are grounded in lived delivery experience and remain the author's responsibility.

Microsoft Copilot has been used deliberately as an assistant in shaping this document - particularly to:

- improve structure and internal consistency
- challenge clarity and intent
- provide objective review
- and reduce friction in turning experience into written form

Copilot did not generate the ideas in this language. It supported their expression. Being explicit about this is intentional.

A language concerned with making implicit things visible should apply the same discipline to how it was created.

On Lineage and Prior Art

This work sits within a broader tradition of pattern-based thinking.

Readers familiar with Christopher Alexander's *A Pattern Language* will recognise the idea of naming recurring problems and forces so they can be reasoned about and adapted rather than enforced.

Those familiar with software pattern literature, including the work of Martin Fowler, will recognise a similar preference for experience-backed patterns over prescriptive frameworks.

This language does not attempt to replicate or extend those bodies of work.

It applies the same underlying principle - that naming recurring structures changes how people think and act - to the specific, bounded domain of ServiceNow delivery.

GET IN TOUCH

If this sampler sharpened the way you think about ServiceNow delivery, keep in touch - I share practical patterns, examples, and hard-won lessons from real projects.

Follow / subscribe

- Philip Swann
  - https://www.linkedin.com/in/philip-swann-vp/
  - https://www.youtube.com/@ValuePattern
  - #ValuePattern

Wrangu

Wrangu are trusted specialists delivering Risk, Security, Governance, and Compliance solutions on the ServiceNow platform for over a decade.

We partner with large organisations across the public and private sector, providing advisory and technical implementation services that bridge the gap between processes and the power of the ServiceNow platform.

With 500+ projects under our belt, a 4.8/5 customer satisfaction score, and clients returning for an average of five projects, we do more than implement solutions. We are expert business transformation partners, delivering long-term value that goes well beyond go-live.

Interested in about our services? Get in touch!
Email: hello@wrangu.com
Browse our services: https://www.wrangu.com

We also share insights from our experts across our channels:

- Subscribe to our newsletter
- Follow us on LinkedIn https://www.linkedin.com/company/wrangu/
- Watch our podcast