Meta Skills

Slide 1 - Title & Intro

Today I want to talk to you about meta skills - what they're, why they matter, and how you can practice them to enhance both your own career and the communities you work with.

Slide 2 - Definition

Sometimes, you'll hear people talk about "hard" skills and "soft" skills, but this is a distinction that I find less useful and accurate than categorizing skills as either "technical" or "meta" skills.

- Technical skills are specific to a domain or role - writing PowerShell, configuring a network, throwing a frisbee, etc.
- Meta skills are the foundation that let all your other skills shine through
- Without solid meta skills, you'll always underperform. That's what this talk is covering in more depth.

Slide 3 - Meta Skills List

Before we dive in, here's the list if meta skills we're going to cover today. I won't give detailed definitions here, because we're going to cover each of these skills in-depth for the rest of this talk. But it's important to note that these skills.

- Apply across all domains - you'll use them in every role and field you work in, as well as your personal life.
- Lifelong improvement - you're never done improving these skills, you'll always be learning more and applying them in new situations and ways.
- Reinforce each other and other skills - you'll find that these skills build on each other and make you more effective at most other tasks.

Slide 4 - Open Source

Today, our in-depth examples are going to use the domain of open source, so we should talk about what we mean by open source for a bit here.

- Most commonly, when people think about open source, they think about software, with the code designed to be:
    - Publicly accessible - free to read on the internet
    - Modifiable - you can copy the code and change it or extend it to suit your purposes
    - Free to redistribute - you don't have to pay royalties to distribute your modified code

Those items are typically covered by the software's license, like MIT or Creative Commons.

As critical as the licensing is, open source is also defined by how it enables collaboration:

- Open source is typically decentralized, the work spread across multiple people and machines, meaning that even when some components aren't available, the whole continues.
- It exhibits and even champions a culture of peer review & community production, leveraging different expertise and availabilities to keep the work moving.

Open source also represents a movement that goes beyond the production of OSS:

- Finds new ways to solve problems, building on the tools and solutions and thinking of those who came before
- Crosses communities and industries - pulling competing companies to standardize better tooling for security, or ensuring there's a shared protocol for a new way to run applications.

Slide 5 - Elements of Open Source

Open source projects are traditionally volunteer driven, without pay. More recently, companies are supporting or hosting open source.

An open source project's community:

- Is made up of people interested in using, supporting, and maintaining the software, not just those contributing code to it.
- Establishes a shared mindset - goals for what to accomplish, values to guide and reinforce in the development and practices, rules to inform how to collaborate.

Project contributors:

- Actively participate, writing code and documentation, testing releases, reporting issues, and proposing new features.
- Can be volunteer maintainers, community members, or paid staff.

Maintainers:

- Are project leaders, responsible for overall health and direction of the project and its ecosystem.
- Define and enforce standards for participation, set priorities for the project.
- Aren't (usually) absolute rulers - they have to listen to everyone else and usually decide transparently and with consensus.

Slide 6 - Systems Mastery

With all that context laid down, let's talk about the first meta skill: Systems mastery.

> Your knowledge of and ability to **exploit** a given system, including its rules, conditions, context, and supporting components.

Systems mastery is often thought of as an achievement, as a thing you gain, but it's also a skill. Learning to explore, understand, and leverage systems is a skill unto itself.

We usually tend to think of systems as collections of hardware or software components, but that's two types of systems. Here are few other things that are systems and subject to system mastery:

- Playbooks, documentation, SOP (explicit systems)
- Workflows, organizational practices, hierarchies (implicit systems)

You can leverage these explicit and implicit systems. For example:

- How do you get changes through a Change Advisory Board (CAB)? Is there someone who has an absolute veto that you need to convince? Does the board completely ignore requests when fields aren't fully filled out? Mastering the system involves learning the ins and outs of the paperwork, the people, and the processes as they actually exist and interact. When you understand them, you can get your changes approved quickly and without rework.
- How do you get budget for training or conferences? When does your organization assign budgets, what criteria do they use? Who decides on prioritization, and how do you make your request a priority for them? How do you describe the value of your work in a way that fits these systems?

Slide 7 - Choosing Projects

Our first example of applying a meta skill is how you can use systems mastery to choose the project or projects you want to contribute to.

- It's easiest to contribute to things you know about or use regularly
- A great way to contribute to projects you're passionate about - you can create a feedback loop where the more you use and contribute to a project, the more you master its systems, and the more effectively you can use and contribute to the project.
- You don't already have to be an expert to gain expertise - you can contribute to a project as a brand new user, providing insight and feedback from a perspective that's notoriously difficult or expensive to retrieve by the project team.

Slide 8 - Communication

The next skill to discuss is communication.

> Your ability to **clearly** and **concisely** convey intent and information, as well as **synthesize** the same from other sources quickly and accurately.

Given that definition, how does this meta skill apply more concretely?

First, the delegation of work and problem spaces is a defining trait of senior skill levels.

- Delegation requires accurate communication of mental models - you can't effectively assign work to someone without making sure they understand what they need to do, why, and how they know if their efforts are successful.
- Accepting delegation requires skill of seeking clarity and confirming - did you understand what was asked of you? Have you explicitly confirmed the acceptance criteria, potential difficulties, and instructions?

Next, consider implementing a new tool - do you know what the users need? How they'll use it? Can you provide useful, clear instructions for those users?

When you're fixing a bug or issue, you need to understand the context - what went wrong? How does it impact users and systems? Who can you reach out to for help? Who needs to know about the issue that might not already be looped in?

Finally, consider handing off a ticket when you're going off shift or taking a vacation - what does your coworker need to know to pick up the work? What do stakeholders need to know about the transfer?

Slide 9 - Filing Issues

For practicing communication in open source, consider filing issues.

The easier it's to understand an issue, the easier (usually) it's to address

High quality Bug reports are hugely valuable to project teams - Rather than a bug report saying "`X` is broken" - which requires the team to communicate effectively with you in a back-and-forth conversation that takes time and energy, filing issues like "I expected `X` to work like `Y`, but `Z` happens. Here's the steps to reproduce the problem..." means that the team can learn everything they need to know about the problem from your report, and they can fix it more quickly and with a higher chance of solving the actual problem.

Similarly, when filing improvements instead of "This project should X" try writing a user story, like "As a `<person>`, I want `<feature>` so that `<value>`..." - this helps the team understand who wants the feature, what the feature should do, and why it's valuable. These are questions they need to answer to prioritize the work. The easier it's to understand and prioritize, the more likely your feature gets worked on.

Slide 10 - Empathy

Next, let's talk about empathy.

> Your ability to develop and understand the mental models of others, *especially* the **feelings**, **perspectives**, and **goals**.

You can't effectively build systems without understanding the people they're for.

- All code is *for* someone. All systems have first, second, and third order effects - not just what we intend directly, but the byproducts of the system and the ripples of those byproducts too.

- Troubleshooting requires understanding where the user is coming from - they're not stupid, they're making decisions and taking actions that made sense to them in their context.
- Our work in operations and infrastructure engineering requires empathy - we need to understand our users, peers, and the broader teams we impact and interact with.
- Justly built systems require thinking about who they affect and how. If we don't think proactively about our systems, we risk building systems that are inaccessible or even harmful, just because we didn't put in the work.
    - For more information than I can fit into this talk, I recommend reading Just Culture by Sidney Dekker, which makes a compelling case for prioritizing empathy in the systems we build, digital and otherwise.

Slide 11 - Conversations in the open

Open source requires us to practice empathy because of the transparent, public nature of the collaboration.

- Working in the open can be stressful - everyone can see your mistakes and how you communicate. Showing patience, seeking clarification, and building both understanding and rapport with people helps you to get through discussions and find solutions. Understanding context drives effective communication - you're often collaborating asynchronously, in text, across cultures and timezones. reproduce? Does the request not fit the project goals? Does the team lack resources? Understanding
- When an issue is closed, it might be for any number of reasons - was it a bug they couldn't reproduce? How the maintainers decide whether to close an issue can help you understand whether you should try an alternative or move on, or whether the issue is open to further discussion. more into this problem later on, but for now you can think about how the team is making those
- Project prioritization is rarely personal, it's usually resource/context constrained. We'll get decisions and what might impact them.

Slide 12 - Collaboration

Let's build on the prior skills even more by discussing collaboration.

> Your ability to work is one ore more other people towards **shared** goals.

The most technically proficient engineer working alone will always be outperformed by 3 mediocre engineers collaborating effectively. You reach a point where you're writing the most lines of code or producing the most new widgets possible in a given hour, and you only ever get marginally more productive on your own from there. When you effectively collaborate, you can make the team overall more effective than the sum of your individual outputs alone.

You can lead from anywhere, leadership != title - peer leadership is just as important to the success of our work. It happens all the time - in any given set of tasks, someone will have more knowledge or experience or be better suited to some of them than the others. We're always arranging and rearranging micro-leadership between collaborators as required by the context we're working in.

We often find ourselves managing individual, team, departmental goals - sometimes they're the same, sometimes they're aligned, and sometimes they're at cross purposes or even incompatible.

The guiding principal at work and in collaboration is doing your best to do right.

Slide 13 - Working through PRs

In open source, you can practice collaboration by working through pull requests (PRs).

The most immediate practice point is when you submit a PR and get feedback, you need to find a way to incorporate it. This is a great way to practice aligning your efforts to the project. At the same time, it requires you to practice empathy, understanding why you're getting that feedback, and how to respond to it.

Reviewing a PR requires you to understand what the change is doing and why - you can't effectively review something without understanding it. Sometimes, seeking that understanding is a critical part of the review itself, because the work isn't done until it can be understood. You'll find yourself asking questions and working to unravel hidden assumptions or missing context.

Adopting PRs is where you take ownership of and accountability for getting an existing proposed change over the line and into the project. This can save an abandoned but valuable change, or free someone else up to do other work, or help someone who gets stuck.

Commenting when you're not a maintainer or the implementer is a great way to provide perspective and value to the project - obvious to some isn't obvious to all. And understanding what's happening is valuable for both current and future users and maintainers. Just be sure to follow the project's practices and code of conduct.

Slide 14 - Resource Management

Resource management is a skill that really helps raise our effectiveness at work and in our personal lives.

> Your ability to **analyze** and effectively **allocate** your available resources to solve problems now and in the future.

- Every project includes folks with varied skills, interests, and availability
- Everyone has more work than they can accomplish
- You need to figure out how best to spend your time and effort to effect maximum impact
- How much time on reporting vs implementing vs testing?
- How do you allocate? Rotate contribution types? Focus? Improve contributing guides?
- This same skill applies, like the others, to any context you find yourself in. We never have infinite resources to tackle the problems and opportunities we face.

Slide 15 - Roadmaps and commitments

Practicing resource management is, perhaps, most obvious when we consider roadmaps and commitments.

- Not every requested feature can or should be implemented - if we had infinite resources, we'd do it all, but that's just not the way it is.
- How to help get a feature prioritized? You could try writing high-quality issues, helping with the work, volunteering to champion an issue, discussing it with the community and team to raise awareness and seek consensus.
- Where are your efforts best spent? What's best for you? For the project? Answering these questions will help you make the most of your limited time, energy, and attention.

Slide 16 - Lateral Thinking

The last skill to discuss is the meta meta-skill, which builds on all the others - lateral thinking.

> Your ability to solve problems by **varied approaches** and creativity.

Lateral thinking applies throughout the lifecycle of working through a problem or opportunity. In operations, things go wrong, and lateral thinking helps us get through it.

- The first question we often ask is, "What went wrong?"
    - Get bug info, try to understand what person was doing when they discovered the problem
    - Practice communication and empathy to build a mental model of what happened and make sure you have the correct information
- How do you investigate?
    - There's a lot of options you can try, like running tests, checking logs and reports, probing the system.
    - Can you narrow things down from broad chunks, then zoom in?
    - This step requires you to apply lateral thinking directly, building on all of the prior skills as you communicate, experiment, manage your resources (including time in an ongoing outage), collaborate with multiple people or teams, and center the needs of the stakeholders.
- How do you fix?
    - Again, you have numerous options, depending on the context and your lateral thinking. Do you roll back? Apply a wuick forward fix now, plan for a long term fix later? Do you restore 90% functionality and write a note about the degraded edge case?
    - Likewise, this requires lateral thinking and builds firmly on all of our other meta skills.

Slude 17 - Seeking alternatives

For the last way to practice a meta skill through open source, consider seeking alternatives when you're contributing.

- Maybe you can do tutorial videos, or blog posts, or answer questions online, or write a plugin, or add test cases, or...
- Does an issue require a code update? Can you document a workaround, or clarify usage, or add an alias, or...
- Sometimes a bugfix for broken behavior indicates an opportunity for enhanced behavior or safety. How can you improve the project beyond just patching a problem?

Slide 18 - Review

- These six core metaskills feed into everything else.
- They're applicable throughout your career, regardless of field
- You can practice them in a low-friction, low-stakes context with open source
- Filing issues, submitting and reviewing PRs, writing docs, adding tests and repros, discussing
- Focus on the skills you want to practice the most - they all affect each other
- Be the contributor you wish would volunteer to help you

Slide 19 - Call to action

Contributing to Microsoft Docs

- PowerShell Contributor Guide - https://learn.microsoft.com/powershell/scripting/community/contributing/overview
- PowerShell Style Guide - https://learn.microsoft.com/powershell/scripting/community/contributing/powershell-style-guide
- PowerShell Editorial Checklist - https://learn.microsoft.com/powershell/scripting/community/contributing/editorial-checklist

PowerShell Community blog

- Blog -https://devblogs.microsoft.com/powershell-community/
- GitHub repo - https://github.com/PowerShell/Community-Blog

Learn site Community hub

- https://learn.microsoft.com/community/

Book recommendations

- Radical Candor by Kim Scott
- The Five Dysfunctions of a Team by Patrick Lencioni
- The Phoenix Project by Kim, Behr, Spafford
- Just Culture by Sidney Dekker