Leading Software Teams with Context, Not Control

Effective practices for creating alignment and engagement within software teams

Dion Beetson

Why This Book was Written

As a software engineering leader, the scope of your role is extensive. You have many competing responsibilities and priorities that need to be balanced to ensure you and your team are as effective as possible. These can include providing architectural direction, driving peer-to-peer collaboration, ensuring cross-team alignment, motivating teams with purpose, supporting team members' career progression, or perhaps, helping remove blockers and impediments. All of these efforts work to create a specific culture within a software team that aims to improve effectiveness, engagement, and retention.

I wrote this book for software leaders who are responsible for leading teams. More specifically, it focuses on approaches for leading multiple software teams, whether directly or indirectly through leadership roles reporting to you. There is a level of unique complexity that comes with leading, aligning, and supporting multiple software development teams; this book aspires to provide you with helpful and reusable approaches that can be leveraged to bring greater efficiency to your role as a leader. There are many books written around leading teams or leading people, but this book provides a lens into the specific practices and initiatives you should be investing your time into when leading software teams.

This book was initially intended to be a 10-part blog post series documenting the practices and initiatives I run while leading software teams. It was to provide myself a model to refer back to and reduce the cognitive load required around the initiatives I needed to orchestrate when leading software teams. As I progressed further into this project, documenting efficiency tips for my own personal benefit, I realised there was value in sharing the approaches I follow with the wider community. This book has been inspired by the many people I have worked with, learnings from initiatives I have implemented, as well as books and blog posts I have read over the years. I see this as a book that lends itself to many iterations and should evolve as I discover new practices and techniques that help me improve the way I lead software teams.

Regardless of the size of your software team, if you find yourself needing to better balance both the technical and people aspects of leading teams or guidance on initiatives you could be running to improve team alignment, effectiveness, and engagement, then this book is for you.

About the Author

Dion Beetson

I am based in Sydney, Australia and have worked in the software industry for the last two decades. I was lucky enough to choose a career right out of university that still motivates me today; I know many people are not this fortunate. I spent the first part of my career focused on fine-tuning my technical skills on all things related to software development. However, I continued to find myself taking on leadership responsibilities within the organisations I worked at. I put this down to being obsessed with solving problems different teams were facing, but also enjoying a sense of satisfaction in helping team members succeed within their task, project, or role. It is an inspiring feeling to observe members within your team succeed.

I've been leading software teams for the last 12 years across a wide range of organisations and industries including media, telecommunications, retail, and wagering. These teams ranged from just a few software developers in a single team start-up to around 150 software developers, testers, and architects in many teams across multiple countries. I've had fun leading large-scale platform migrations, and have experienced the stress of scaling high-throughput applications to meet the daily needs of tens of millions of customers.

Like most software developers, there is a point in your career when you begin to consider how far you want to progress into leadership compared to hands-on software development. For me, this occurred at two distinct points in time. The first was when I took on the responsibility of leading four developers, a role I naturally progressed into. Due to the juggling act between software development and leadership, I worked longer hours than most to ensure I could execute both competently. This role also provided a safety net of moving back to a purely software development role if I so chose.

The second transition into leadership was the hardest and occurred over two years within two roles at two different organisations. This was when I had to make the tough decision to step back from coding in my day-to-day role and focus entirely on people and technical leadership. This was challenging, as I knew it would be hard to easily move back into a software development role quickly. However, I ended up taking the leap and haven't looked back since.

I chose to refocus my career on leading software teams for a variety of reasons, but one stands out more than the rest. I have experienced many times where years of great software development work ceases to exist and platforms or projects are shut down due to variables outside of a team's control. This is rarely due to malicious reasons; in many instances, it is simply the result of cost cutting, acquisitions, or an organisational pivot in another direction. Regardless, it is a soul-crushing outcome and really made me contemplate how I could better focus my efforts. I was driven to find a way my hard work could last through these unfortunate situations that were destined to reoccur throughout the rest of my career, whether I liked it or not. The solution was to focus on building software team cultures that would motivate individuals to rally behind and be part of. With enough support behind a software team culture, it will continue on well after you leave - even after an organisation ceases to exist. If individuals believed in the culture you created and how you led them, they will take that mindset to their new organisation and, hopefully, have a positive impact there. This is a truly inspiring outcome.

Leadership, however, is an entirely different career pathway, and although having a software development background definitely helps you understand technical teams better, you need to start from the beginning and build up a new set of skills and experience. I was fortunate enough to work with many amazing software leaders who helped guide me through this transition, each in their own way. 'Don't lose the technical side, Dion. That balance is what makes you valuable', was a powerful piece of advice from a past manager I will always remember, and an opinion that was commonly reinforced by other leaders I respected and worked with. What it represents is if you can find a way to balance leading teams and keep your technical skills up to date, you have a considerable advantage over peers within the industry.

As team sizes grow, most senior software leadership roles will require little to no coding within their day to day. But you still have an obligation to your software team to understand their architecture, constructively challenge their approaches where needed, and support them both technically, and as a leader. There are many approaches to staying up to date technically when in a leadership role. These can include personal software projects after work, reading articles and books, or proactively getting involved and conversing with peers on technical solutions. Either way, to stand out, you shouldn't fall behind on software engineering principles if you want to directly lead software developers. An important disclaimer is that everyone will have their own journey and decisions to make. Ultimately, you need to decide on what motivates you, what can you do every day with that same motivation, and which role continuously drives you to proactively learn how to master it better.

To summarise my career aspirations, it is to be recognised as a software leader who has a reputation for building software teams that people positively remember being part of, where they were technically challenged, where they owned their decision-making, and had an opportunity to advance their career while still adding value to the wider organisation. I focus on context, not control, in that I work hard to ensure my software teams have the context they need to make and own decisions in their entirety within the team without me.

1. Baselining a Software Team

Stepping into a software leadership role within a new or existing organisation presents endless opportunities to deliver improvement. These possibilities of uplift can drive you to wake up every morning ready to start the day more motivated than the last. In most instances though, your understanding of the team's current state and aspirational state will be minimal at best. This is due to only being exposed to a few hours of discussions and context throughout the interview process. The interviews may have provided you with a simple understanding of the technologies in use, team structures in place or software engineering practices the team is aspiring to follow; however, in-depth knowledge to make informed strategic technical decisions will seldom exist. For example, how do teams tackle dependency problems? What is the team's observability posture? What part of the team's deployment processes are most bottlenecked? How do teams' measure platform uptime? Or more importantly, what are the current uptime metrics of each platform?

One of the first exercises within your new role is to baseline the team. This will enable you and your entire software team to understand the current state of play and allow improvements to be iteratively introduced over time. Baselining a software team needs to focus on a range of technical and cultural measures, and it is your responsibility to work with the team in defining what is important. This may include industry standard measures like speed of production deployments, the health of the team's technical debt backlog, or clarity of the software team's target state aspirations. There will also be measures that are specifically important to your team, perhaps a services ability to self-heal to reduce customer impact, the availability of reference architecture documentation, or the platform's observability maturity.

Collating baseline measures is led through your role, but is considerably more valuable when software leaders amongst the team collaborate. This will ensures a balance of measures from different perspectives and experiences are taken into account.

Why Baseline a Team?

Having led software teams, you have undoubtedly found yourself in a situation where you too quickly defined and implemented a strategic decision that resulted in a less than desirable outcome. As a software leader, understanding the current baseline of a team before contemplating introducing change is fundamental. In a survey conducted by HBR that gathered feedback from more than 50,000 leaders around behaviours that led to poor decision-making, laziness was at the very top of the list. Laziness in checking facts, confirming assumptions, and in our case, understanding the current state of a software team and their platforms before making key decisions. The survey also identified decisions that were made in isolation by leaders without engaging their team or peers resulted in a reduction of quality in the outcome of those decisions.[1] As a software leader, it is important to understand where current problems are and why those problems exist. If you fail to understand the why, you increase the risk of introducing change that may suffer from the same challenges you are ultimately trying to resolve.

Once a team's baseline has been clearly understood and documented, it opens up the opportunity to focus on a range of initiatives that enable levelling up the team. These can include defining what software excellence represents to encourage an aspirational target state, development of a software roadmap to facilitate alignment, or definition of team goals and metrics to create focus and accountability. Chapters 2 through 5 focus on these initiatives, which I recommend implementing sequentially, as their inputs feed on the outputs of the previous. Later chapters explore further initiatives which can be implemented in parallel, like how to balance reactive versus strategic priorities (Chapter 6), leading teams with context not control (Chapter 17), as well as developing career pathway frameworks (Chapter 16) to provide growth opportunities to individuals.

All of these baseline measures support in driving a specific culture within your software team. Each team's culture is unique to them. One team might be focused around empowering end-to-end team ownership of platforms, while another may be focused on a transformation where coaching and learning play a vital role. It is important to recognise that many software teams will have a blend of different measures that define their culture; and that is completely OK. There is no one-size-fits-all culture definition for a team. Baselining a software team's current state is the first step to opening up initiatives to focus on next. Neglecting the current state is the equivalent of developing a new feature in an existing codebase you have never worked on before, without first understanding the capabilities that exist within that platform.

How to Baseline

There is an endless list of measures that can be used to baseline a software team. It is useful to initially cast a wide net to make sure key measures are not overlooked; however, it is more important to ensure the final consolidated list includes only those measures that will continue to be relevant to the team over the next two to three years. The reason is that driving change, especially in larger teams, will take many years. Each team adopts change at a different pace due to their maturity, their make-up and their need to balance day-to-day work efforts. Limit yourself to a maximum of 10 baseline measures that you and your leadership team agree are the most important in driving a high-performing software team.

Defining What is Important to Your Team

The first step to baselining a software team is to define the measures the team believes are most important. Dividing this into two themes, technical measures and cultural measures, will provide structure and boundaries to frame the conversation.

This allows you to build out a fitness matrix for technical measures relevant to the team's platforms, and then a separate fitness matrix for team cultural measures. If needed you can choose to define different or even additional themes that are relevant to your team. Just remember the more themes there are, the more measures there are, which means the process to baseline will require more effort. As a leader, it is important to treat your time as a limited and precious resource. I recommend starting small then iterating in future months as needed, just like we would when developing a software capability and continually iterating on it even after it has been deployed into production.

Although brainstorming measures can be done in isolation, there is a considerable upside to encouraging a collaborative approach with key leaders within your software team. Collaboration builds trust in relationships which is fundamental to you and your team's success, especially when entering a new organisation.

Exercise:

- Set aside 45 minutes to run a session on brainstorming baseline measures.
- Invite relevant software leaders within the team which may include team leads, senior software developers or software development managers.
- Provide each attendee with the expectation to come prepared to share at least one recommendation for a technical measure and one for a cultural measure.
- Within the session, place two headings on a wall, one for technical measures and one for cultural measures.

Technical measures	**Cultural measures**
…	…
…	…
…	…

- Provide each person a handful of Post-it Notes.
- Allow five minutes for each person to silently refine and write down at least one technical measure and at least one cultural measure. Then put them all on the wall under their respective heading.
- Spend five minutes grouping together similar measures.
- Allow 10 minutes for each person to individually read through each Post-it Note on the wall and vote on their top five measures by placing a single dot on each Post-it Note they vote for.
- For the top 10 voted measures, go through each item and discuss for two to three minutes on 'why' that measure is important to the team. Once agreed, write it onto the Post-it Note and decide whether to keep or discard the measure if no longer relevant.
- By the end of the session, the objective is to have 10 or fewer measures that align with both the technical and cultural themes.

Reference of Baseline Measures

Below is a list of technical measures that are common amongst software teams. You can use these as discussion starters or to seed your team's baseline measures.

Measure	Measurement	Why it is important
All code is in a VCS with an agreed branching model	Yes/No	Using VCS creates a single source of truth, while an agreed branching model helps enforce peer reviews and safer deployments.
Secrets not stored in VCS	Yes/No	Storing secrets in VCS is a critical security vulnerability that needs to be avoided at all costs.
Infrastructure is as code	Yes/No	IaC encourages repeatability and reusability of infrastructure.

Measure	**Measurement**	**Why it is important**
Time to roll forward or roll back a release in production	Minutes	The ability to roll forward or back quickly means a team can also deploy single features or fixes quickly. This leads to safer and lower risk releases. It also provides confidence that in the event of a failed release, outage times are reduced which provides a better customer experience.
Automated build pipelines exist	Yes/No	Automated build pipelines create a consistent path to production that reduces risk and potential for human error.
Consistency between non-production and production environments	% of consistency between environments	Ensuring consistency between non-production and production environments reduces edge cases and defects in production.
Centralised logging of application error and access logs	% of application logs centralised	Centralised logging allows for faster debugging and thus faster recovery times in the event of a failure.

Measure	**Measurement**	**Why it is important**
Observability of critical services	% of critical paths monitored	If a platform is not monitored, it cannot be alerted on. If a platform cannot be alerted on when a critical issue or degradation is occurring, it may result in an outage and poor customer experience.
After-hours support	Yes/No	Not all platforms require this, but critical platforms should have an after-hours support framework to reduce the need to chase people outside of business hours to resolve production issues.
Documentation and reference architecture exists	Yes/No	Does the platform have a reference architecture documented that explains why the current architecture was chosen? This assists in onboarding new team members. The objective is to find a balance of 'just enough' documentation to support the team.

Measure	**Measurement**	**Why it is important**
Self-healing and graceful degradation of services	Yes/No	Are the critical paths within services able to self-heal or gracefully degrade to reduce customer impact?
Healthy technical and product backlog	Yes/No	Does the team have a healthy backlog to balance technical uplift and product work?
SLAs exist	Yes/No	Each platform needs defined SLAs that customers measure them by and that the team can then measure themselves by. It is imperative that any SLA is automatically measured with zero operational overhead.

To complement the above technical measures, the following is a list of cultural measures that are also common amongst software teams.

Measure	**Measurement**	**Why it is important**
Teams own their technology stack	Rating 1-10	The more a team can own their technology stack, the more it reduces dependencies on people outside the team which supports faster decision-making. Ownership encompasses architecture, development, testing, releasing and supporting a platform.
Efficient onboarding experience	Rating 1-10	Teams need to have a supportive onboarding process to bring new starters up to speed efficiently. This is especially important when teams are actively recruiting. Chapter 14, Effective Onboarding, discusses approaches to onboarding within software teams.

Measure	**Measurement**	**Why it is important**
DevOps/SecOps/*Ops mindset	Rating 1-10	This *Ops mindset increases early collaboration as it shifts left much of the thinking around security, infrastructure and QA. This mindset is hard to find when recruiting and even harder to introduce.
Motivating team purpose exists	Yes/No	Without a purpose for existence, teams will struggle with long-term motivation.
Target state architecture defined	Yes/No	A defined target state architecture ensures every member is aligned in the same direction.
Team composition is correct	Yes/No	Teams that have the correct composition of technical skills and personalities are higher performing teams.
Key-person risks exist within the team	Yes/No	Key-person risks impact the team’s efficiency when individuals move on.

Measure	**Measurement**	**Why it is important**
Teams solution together	Rating 1-10	Teams that solve problems together collaborate better and share knowledge more effectively.
Team members are remunerated fairly	Yes/No	Remunerate people what they are worth and no less. This takes money out of the equation and supports them in focusing on being a valuable member of the team who delivers long term value.

Bringing it Together

Once baseline measures are defined and agreed upon, the next step is to use these measures to assess the current state of the software team. One simple approach is to list all of the measures as rows, and all of the platforms or teams within your organisation as columns to form a matrix. This is represented over two individual tables based on technical and cultural measures.

Technical Baseline Measures

Measure	**Platform 1**	**Platform 2**	**...**
Time to roll forward or roll back a release in production	[rating]	[rating]	...
Observability of critical services	[rating]	[rating]	...

Cultural Baseline Measures

Measure	**Team 1**	**Team 2**	**...**
Teams own their technology stack	[rating]	[rating]	...
Efficient onboarding experience	[rating]	[rating]	...

With a software team's baseline defined and measured, this plays a pivotal role in defining future improvement initiatives the team will take on. It is important to realise not every improvement can be worked on simultaneously - improvement is a continuous journey. Teams need to find a level of capacity to focus on improvement throughout their day-to-day work that is balanced with other initiatives. Teams should analyse and identify measures that return the highest value for the least amount of effort and focus on these items first. This is commonly referred to as the low-hanging fruit.

A software team's baselines should be continuously measured and where possible, automated. In the upcoming chapters, subsequent initiatives like target states (Chapter 2), road mapping (Chapter 3) and goal setting (Chapter 5), which all aim to uplift a software team's baseline, is discussed.

TL;DR

- Baselining a software team needs to focus on a range of technical and team cultural measures. It is your responsibility to work with the team in defining what is important.
- Understanding the current baseline of a software team is a fundamental necessity. Otherwise, you risk introducing change that may suffer from the same challenges you are ultimately trying to resolve.
- Start with a minimal set of baseline measures, then iteratively add to them as and when needed, within the team.

10. Effective 1:on:1s

1:on:1s are weekly (preferred) or fortnightly catch-ups with each of your direct reports. They provide a dedicated time when you, as their manager, are 100% accessible and focused on supporting them in their role. It is an opportunity for you to listen, provide guidance, listen even more, and collaborate on challenges or problems they're facing. The conversations within 1:on:1s can move in many different directions around career, present challenges, current or new initiatives, blockers or even feedback. However, at the absolute centre of every 1:on:1 is coaching and mentoring. An essential aspect of your role as a leader is to challenge your direct reports to continually improve in an effort to support them advancing in their career aspirations.

There isn't a single approach you should use to orchestrate 1:on:1s consistently for every one of your team members. Each individual is unique, so naturally, 1:on:1s should evolve based on their personality and the relationship between you. These 1:on:1s should be engaging for both parties and not result in a status update meeting; status updates can be easily gathered through quick pulse-check meetings, email updates, or chat messages. This isn't to say that you should avoid status updates at all costs. There is value in a portion of a 1:on:1 focusing on status updates as your direct report may be excited to share specific progress with you. Just remember, status updates should be bidirectional and you shouldn't be going into a 1:on:1 with the sole purpose of extracting updates.

Common expectations are that 1:on:1s should be entirely led by the direct report, and that the direct report should come prepared with input on new technical direction, their career aspirations, and how to resolve current challenges. In some fortunate situations, you may find yourself with high-performing and proactive direct reports, however, that doesn't mean you can expect them to lead every 1:on:1. You also need to come prepared to each 1:on:1, just like you expect them to. This might be with insightful questions on how they are planning to progress to the next step within their career, or it might be a feedback conversation around how they could have handled a recent situation more constructively. Sometimes, it may just be coming prepared with a conversation starter that encourages a quiet direct report to open up. Great discussions can often eventuate from questions like “Where would you like to see more direction within the software team?”, or “what is the biggest challenge that keeps you awake at night?”.

The Value 1:on:1s Bring

Above all else, meaningful 1:on:1s assist in building trust between the two of you. Trust is fundamental for any relationship to collaborate and deliver together. As a software leader, you build trust by showing empathy, by being genuinely interested, by giving honest feedback, or by supporting them in their career progression. A 1:on:1 is not just a weekly meeting you run because it's what managers do - you run it passionately because it provides you with fascinating insights around what is motivating or frustrating your direct report. These meetings allow you to indirectly learn about potential risks or problems before they turn into disasters that disrupt the team.

Perhaps the most enjoyable aspect of 1:on:1s for me is being able to listen to the challenges my direct reports are facing and act as a sounding board for different approaches to improve the situation. These challenges normally relate to conflict between team members, misalignment in the direction we thought was clear, and how to better support an underperforming team member. It is important, though, to not solve the challenge for them, but to share your experience in the matter, talk to approaches you might take, and guide them in a direction that allows them to solve it for themselves and learn a new experience along the way.

Lastly, there is absolutely nothing wrong with a 1:on:1 being simply a coffee chat that revolves around conversations entirely unrelated to work. These might be about weekend plans, upcoming holidays, family topics, or discussions about a recent article either of you read. Sometimes, these conversations can be just as important as work-related ones, as they act as an outlet and a way to boost an individual's mental energy to help them better tackle upcoming challenges within their day-to-day responsibilities.

Running 1:on:1s

I plan out a fairly structured high-level 1:on:1 format. Although quite detailed, its primary objective is to provide topics I can pick and choose from that are relevant for a specific 1:on:1, but also allows me to keep each one very conversational. This approach helps reduce last minute planning and the cognitive load required each week to run seven 1:on:1s that range from 30-60 minutes. Below is the high-level structure I follow.

1. Always run 1:on:1s outside of the office, whether that is at a coffee shop or even just a walk and talk around the local park. Running 1:on:1s outside the office changes the dynamic in a positive way, allowing individuals to open up more than they would inside of a meeting room.
2. Start off by thanking them for something they have done over the last week.
    - Did they help drive an initiative you asked them to?
    - Did they step up and resolve a cross-team collaboration issue?
    - Did they make a new starter feel welcome?
    - As you can see, it isn't hard to find something to thank someone for, and it goes a long way in building motivated team members.

3. Ask a meaningful question to break out of the context you were both in before the 1:on:1. Any question other than “How is everything going?” is a positive start. For example:

Question	Value the question adds
What has been taking up most of your headspace lately?	Useful when they seem under pressure or stressed.
Where have you been making the most impact?	Useful for finding out if they understand where value is being created from their work. If required, it also allows you to provide input or realign where time is being spent.
What have you been driving improvement in recently?	This is a great question to find out where their motivation is to improve their team.
Where have you or your team been doing impressive technical uplift lately?	This is a good pulse-check question, as if teams are not uplifting tech or reducing technical debt they will end up with unmaintainable platforms that are slow to iterate on.
Where are you seeing platform stability issues?	Useful for finding out stability pain points as well as being a potential sounding board for how to improve stability.
What do you feel is going well within your role or team?	Useful if they seem down or have had a challenging week. Focusing on the positive can allow individuals to see a different perspective.

Question	**Value the question adds**
What is making you nervous about your role or team?	Useful in starting a conversation around where they are potentially unhappy or unable to resolve a problem.
What technology decisions do you feel are important to make at the moment?	Useful in understanding where their interest or passion is. Remember, part of your role is upskilling your team and succession planning. If you are able to align an individual's interest to yours or your organisational goals, that is a win-win situation.
What changes in the software team's direction would you recommend?	As a leader, you don't have all the answers. A lot of the time, your direct reports can provide direction to greatly improve the wider team. You just need to ask them.

4. Ask the question, "What would you like to discuss today?"
    - This open-ended question provides your direct report an immediate opportunity to share important topics they want to talk through. Depending on the individual and the time available, it can sometimes be useful to ask them to quickly summarise their key topics so you can both focus on the important items first.

5. Cascade relevant or useful information.
    - Around half way through the 1:on:1, I will usually cascade recent information specific to them that I don't plan on cascading in a team meeting. This could include upcoming team changes, new technology initiatives, adjustments to the software engineering team direction, or even visibility on initiatives I am currently focusing on.
6. Provide necessary feedback.
    - Feedback, whether positive or constructive, always needs to be shared to support growth within their role. Feedback might relate to:
        - An area they should focus on improving over the next few weeks.
        - Discussion around how they could have handled a recent situation more constructively.
        - Feedback you may have received from other individuals or through skip meetings.

7. Check in on the key areas important to you or the organisation.
    - This can be defined as the status update portion of the 1:on:1. It is valuable to talk through status updates important to the organisation from both sides. This helps your direct report understand the importance of different priorities they are working on. This may be around technical decisions they are making, progress on recruitment, or how a key project they are leading is tracking. I attempt to limit this to no more than 20% of the 1:on:1.
8. Check in on role or career goals.
    - Within the first 90 days of starting a new software leadership role, it is important to understand the career goals of all direct reports. Following that, work with them to identify specific milestones they are focused on achieving to move their career forward. This question allows you to check on how they are progressing with these specific career objectives. Having just one person like their manager show genuine interest in how they are progressing in their career goals can go a long way in motivating them to actually achieving them.
9. Where are they wanting support over the next week?
    - Finishing off every 1:on:1 with an open-ended question in where you can support them over the coming week allows them the opportunity to ask for help.

It is not necessary or recommended to talk through every one of the above topics in a single 1:on:1 due to the time it would take. It really depends on the individual, what is important to them at that current point in time, and how many discussion items they bring to the session. Splitting these topics across two or three 1:on:1s can usually provide enough time and space to dive into meaningful conversations.

Fifteen Additional Helpful 1:on:1 Questions

1. What is a specific goal you are focusing on to enhance your career growth within the next three months?
2. What change would you be motivated in driving over the next month?
3. If rated 1 out of 10, what is your current enthusiasm level in your role? Why this rating?
4. If rated 1 out of 10, how clear is technical direction? Why this rating?
5. What is blocking or slowing you down the most recently?
6. Who are your key-person risks and flight risks within your team?
7. Where would you like to see more direction within our team?
8. What can we take off your calendar this week to reduce stress?
9. How clear are you on your own priorities?
10. What is one thing you would like to get out of our 1:on:1s?
11. What would you like me, as your manager, to focus on?

12. If looking for quick wins that add value, what would you recommend?
13. Where are we struggling in terms of team communication?
14. What engineering practices do you want to see introduced?
15. What is one skill you want to improve on over the next three months?

Preparing to Ask the Tough Questions

If you are planning on asking a tough question or focus on a specific topic within a 1:on:1, it is in your best interest to provide a heads up a few days in advance. This allows your direct report to prepare, ensuring you have a constructive conversation about it. Without forewarning, it can catch them off guard, which results in a less meaningful and thought out response.

Tackling Negativity Within 1:on:1s

You may be in a situation where a certain direct report demonstrates an attitude more negative than a positive, or even uses a 1:on:1 to continually complain about everything that's wrong. If this is a one off, they may just be having a bad day and you can support them by understanding the problems and working through options to improve the situation. However, if negativity is becoming a trend in their 1:on:1s, it needs to be resolved. There are many reasons for negativity including conflict with other team members, lack of career growth, working on uninspiring initiatives, lack of purpose, or even misalignment in direction within the team or wider organisation. Your role as a software leader is to work with them to understand the root challenges and help them help themselves in implementing a solution. Ignoring trends in negativity will not only become mentally exhausting for you, but will often result in their continued lack of motivation and eventual resignation.

Focus on turning negative statements into constructive conversations that can assist in solving the root challenges. For example:

- If your direct report is continually making broad, negative statements, ask:
    - What do you believe is the root cause? 'The Five Whys' can be a useful exercise to explore deeper.[1]
    - Follow up by asking what is within their control to change that may improve the situation.
    - If they are unsure, ask them to come to the next 1:on:1 with at least one possible change they could implement.
- If your direct report has complained about the same issue over multiple 1:on:1s, ask:
    - What are you doing to improve the current situation?
    - If they haven't acted, be a sounding board on possible tactics, but ensure they own the action and implementation.

- If your direct report is complaining about another team member, ask:
    - Why do you think this person is acting the way they are?
    - What impact is this having on your role?
    - How could you support them in their role without doing their role for them?

Sometimes, negativity and frustration are simply due to their interaction with an underperforming team member. In that situation, the underperforming team member needs to be supported in uplifting and fulfilling their role, otherwise, they should be put on performance management.

TL;DR

- 1:on:1s are weekly or fortnightly catch-ups with each of your direct reports that provide an opportunity for you to listen, provide guidance, coach, listen more, and support them within their role and future career aspirations.
- Meaningful 1:on:1s, above all else, assist in building trust between you two. Trust is fundamental for any relationship to be able to collaborate and deliver together.
- Plan out a high-level structure for running 1:on:1s that allows you to pick and choose different topics to discuss, as well as reduce the cognitive load required each week.