

PharmaSUG China 2026 - Paper LS-221

Leading Through Complexity: A Compound Lead Perspective on Planning, Coordination, and Quality Control

Fuyuan Pang, Mochao Gao Hengrui Pharmaceutical

ABSTRACT

In complex clinical trial settings, the Compound Lead role extends well beyond managing programming deliverables. It requires proactive planning, cross-functional coordination, resource balancing, and practical quality leadership. Drawing on lessons from a recent project, this presentation shares a leadership perspective on how statistical programming leads can operate effectively in demanding study environments.

INTRODUCTION

In many clinical development programs, the statistical programming lead is expected to be the person who makes the deliverables appear on time. That expectation is understandable, but it is incomplete. In a complex NDA project, especially one with multiple linked deliverables such as a pivotal Phase III study, PopPK analysis, ISS support, and safety reporting for DSUR, the programming lead is also a coordinator, a quality guardian, and often the person who holds the most practical knowledge of how the study actually runs.

This paper uses the term Compound Lead to describe a programming lead who is responsible not only for a single deliverable stream, but for the overall progress of a compound set of study activities. The term is meant to reflect both breadth and responsibility. The Compound Lead may not own every analysis or write every production program, but still carries accountability for whether the project as a whole moves forward in a controlled way.

The main idea in this paper is simple: good programming leadership is not passive execution. It is active ownership. A junior statistical programmer can still act as a true lead if that person understands the project, understands the people involved, and behaves as the person most committed to making the work succeed. Leadership in this setting is less about title and more about judgment.

The following three sections discuss how that judgment shows up in practice. Planning focuses on how to allocate work and protect momentum under tight timelines. Coordination discusses how the lead should engage with other functions and translate between project needs and programming realities. Quality Control explains how to build safeguards when team experience and domain understanding are uneven.

PLANNING

A complex NDA program usually does not present itself as one neat stream of work. It is a set of interdependent activities, each with its own timeline, dependencies, and risk. In the project that motivated this paper, the program included a major Phase III study, PopPK work that needed clean data for modeling, an ISS that depended on coherent cross-study integration, and a DSUR that had to be supported on a regular reporting cycle. The study indication also created a tight operational window: because follow-up after enrollment was not long, the time from LPI to data freeze, blinded data review, database lock, topline, and subsequent downstream deliverables was compressed.

That kind of schedule leaves very little room for improvisation. A programming lead cannot wait until the first bottleneck appears before thinking about resources. By the time the project reaches blinded review or database lock, the workflow from SDTM to ADaM to TFL development, integration, review, and validation should already be in motion. If the lead waits too long, the project can stall at the exact moment when the team has the least flexibility.

The first planning responsibility is therefore to estimate workload early and honestly. Every deliverable

should be mapped to its inputs, review cycles, expected revisions, and dependencies on other functions. A simple list of output tables is not enough. The lead must think through how many programming iterations may be required, which outputs are likely to be high risk, where reviewer comments may concentrate, and which activities can realistically overlap. For example, when a topline timeline is fixed, the lead should know which listings and figures are absolutely critical for the first pass and which items can follow after the initial release.

Planning also requires a realistic view of the team. It is not enough to trust colleagues in the abstract. Trust is strongest when it is informed by knowledge. The lead should understand each team member's prior project experience, technical depth, speed, communication style, and comfort with different types of work. A programmer who has only recently learned SDTM concepts should not be assigned a highly complex endpoint derivation that depends on nuanced clinical interpretation without support. That does not mean the programmer should be protected from all challenge. It means the assignment should be challenging in a controlled way.

A strong Compound Lead uses the project itself to develop people. If one colleague has mainly worked on Phase I or Phase II safety analyses, that person can be given smaller pieces of efficacy programming such as line plots, basic efficacy listings, or selected summary outputs. The point is to match the task to the current skill level while still creating an opportunity for growth. Over time, a well-managed project can become a training ground rather than just a delivery machine. That is valuable both for the organization and for the individual programmer.

The risk of poor planning is not only schedule delay. It is also avoidable frustration. If a less experienced colleague is given too much work without enough support, and the work then falls behind, the first reaction is often to think that the colleague failed. In reality, the more fundamental problem is that the project was not planned around actual capability. The programming lead owns that failure because the lead is the one who should have seen the mismatch earlier.

For that reason, the lead must also plan for support, not just assignment. A good plan specifies not only who does the work, but also who reviews, who answers questions, and how escalation will happen if something is unclear. When the team knows that support is available, they are more likely to ask questions early, which is exactly what the lead wants in a fast-moving project.

Planning is therefore a leadership function, not an administrative one. It is about creating momentum before urgency peaks. The goal is to keep the project moving in a way that is both efficient and humane.

COORDINATION

If planning is about setting the project in motion, coordination is about keeping it aligned as the work unfolds. In theory, statisticians may define many of the rules, while programming executes them. In practice, the programming lead often becomes the person who sees how those rules behave against real data. That is why the lead should not think of the role as a passive receiver of specifications.

Real data reveals gaps that specifications alone cannot always expose. A derivation may be technically sound on paper but still fail to produce the intended output structure. A collection rule may look complete until edge cases appear in actual subject data. A review plan may seem adequate until the timing of upstream data changes forces the team to rethink dependencies. These issues are easiest to identify when the programming lead is actively engaging with the work rather than waiting for a formal problem to be handed down.

The Compound Lead should therefore act as the practical connector between project requirements and data reality. That means asking questions early, especially when a rule appears ambiguous or when the downstream impact is unclear. It also means proposing better ways to collect, structure, or present data when the current design is unlikely to support the intended analysis. In other words, the lead is not only preserving specifications; the lead is improving the path from specification to result.

This is particularly important in NDA programs where multiple functions must stay synchronized. Programming needs to communicate with data management on data issues, with medical on interpretation, with pharmacology on model inputs and outputs, with statistics on analysis intent, and with external review groups when independent data review or medical writing support is involved. The lead may not be the formal owner of all those discussions, but should be the person who knows when they need to happen and what programming questions need to be raised.

One practical leadership lesson is that the most knowledgeable person in a project is often not the most senior person in the room. If the Compound Lead is the one closest to the data and the deliverables, then that person should be prepared to answer project-specific questions with confidence. Even when a senior colleague or more experienced programmer is involved, the lead should still own the study knowledge needed for day-to-day decisions. Within the project, the lead is the bridge. Other programmers, vendors, and cross-functional partners may rely on that bridge to understand what is possible, what is risky, and what is ready.

That bridge role matters because it shapes team confidence. When the lead can explain the logic behind a derivation, the reason for a formatting choice, or the consequence of a timeline change, the rest of the team can move with greater certainty. When the lead cannot explain those things, the project becomes dependent on guesswork. Leadership, in this sense, is not a separate layer above the work. It is the ability to translate the work accurately across boundaries.

Coordination is also where the lead can improve the project culture. If a team member raises an issue, the lead should treat that issue as a signal that the workflow is working, not failing. Problems should be surfaced early and discussed openly. The best cross-functional projects are not those with no issues; they are the ones where issues are visible early enough to fix them without drama.

QUALITY CONTROL

Quality control in statistical programming is often treated as a final step, but in practice it should be embedded throughout the process. For a Compound Lead, quality is not achieved by personally checking every line of code. It is achieved by designing a process that makes errors easier to catch and harder to propagate.

In a complex project, especially when the team has mixed levels of experience, the main quality challenge is not just technical correctness. It is consistency of understanding. Two programmers can produce outputs that look similar but differ in subtle, important ways. A listing may be formatted correctly but still miss a critical cross-check. A derivation may follow the specification but still fail to reflect the intended clinical meaning. The lead must understand the outputs well enough to recognize these differences.

One effective approach is to create structured review mechanisms based on clinical trial experience. Instead of reviewing outputs one by one in isolation, the lead should connect tables, listings, and figures to one another and check whether the story is internally consistent. For example, a subject count in one table should reconcile with the denominator used in a related figure. A shift in safety data should be visible in both the subject-level listing and the summary output. An endpoint result should align with its supporting derivation and its source data pathway. These cross-checks are more powerful than isolated visual inspection because they test the logic of the entire output set.

Technology can help here. AI-supported review agents can be useful when they are built around concrete programming rules and clinical context. Such agents should not replace the lead's judgment, but they can automate repetitive review tasks, identify unusual patterns, and flag mismatches for human review. In a time-pressured environment, that matters because repeated batch runs consume time and fatigue increases the chance that a human reviewer will miss something obvious. An agent that checks consistency across tables can help focus attention on the highest-risk areas.

Still, the tool is only part of the solution. The way the lead communicates quality issues is equally important. When a problem is found, especially with a junior colleague, the first instinct should not be to assign blame. If the lead responds with frustration or accusation, the colleague may become defensive and less willing to bring up future issues. That is the opposite of what the team needs. Internal problem

discovery is valuable. It is always better to find a mistake before submission than to have a regulator or external reviewer find it later.

The more effective approach is to ask why the issue happened and what knowledge gap, process gap, or assumption gap allowed it to slip through. Then the lead should guide the colleague to inspect the result again and try to identify the problem independently. This is both a technical and a developmental step. It builds ownership, and it teaches the colleague how to think about similar problems in the future.

There is also a psychological reason this approach works. People are naturally more likely to trust their own output than someone else's. That can make self-review difficult. A fresh reviewer who understands the subject area but did not participate directly in the work can be extremely effective because they approach the output with less attachment. The questions they ask often prompt the original programmer to re-examine the logic and discover a missed issue. In that sense, review quality improves when the reviewer has enough domain knowledge to be relevant but enough distance to be objective.

The lead should use this reality as a design principle. Pair review, structured checklists, and independent cross-checks are not signs of weakness. They are safeguards that make quality more objective when judgment varies across contributors. When those safeguards are in place, less experienced programmers can work with greater confidence, and the team as a whole becomes more resilient.

At a deeper level, quality control is also a leadership signal. When other programmers know that the lead will carefully review critical outputs, explain issues clearly, and stand with the team during problem solving, they feel safer taking on difficult work. That safety improves the project because people ask better questions, report issues earlier, and learn faster.

CONCLUSION

A Compound Lead in a complex NDA program does much more than track deliverables. The role requires early planning, active coordination, and systematic quality control. More importantly, it requires the judgment to connect those three activities into one leadership practice.

Planning creates momentum before the project becomes urgent. Coordination ensures that programming remains aligned with data, clinical, statistical, and operational realities. Quality control protects the final result while also developing the team through structured review and constructive feedback. Together, these practices allow the lead to deliver not only code and outputs, but also a stronger team and a more reliable process.

The central lesson is that programming leadership is not passive. A lead should not wait to be invited into the important conversations, should not assume that specifications will remain correct without challenge, and should not treat quality as a final checkbox. The Compound Lead leads by staying close to the work, understanding the people involved, and taking responsibility for the full path from planning to delivery.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Fuyuan Pang
Hengrui Pharmaceutical
15601795613
fuyuan.pang@hengrui.com