## 36 steps to success as technical lead

The "*tech lead*" role can be treacherous at times. While the name implies "*leadership*", most of the times it doesn't come with implied authority like a manager role for example. It often happens that this role is in a no-man's-land where it brings a lot of**responsibility** but not enough **formal authority**. In order to successfully help a project from this position one has to navigate through narrow and convoluted straits.

*The role is not clearly defined in most companies and it is placed in a continuum starting at the senior programmer level and extending to architecture and management positions.*

More often than not the "*tech lead*" shares the responsibility for a project without being given full formal authority. In this kind of situation success will be based on the ability of the "*tech lead*" to combine his "*tech*" and "*lead*" talents in a manner that will get results from the team and will get approval and support from the management and the business.

I assume you are new at technical leadership. You are perfect for the job *from a technical point of view* but you lack experience in leading others while bearing the responsibility for their work. There is no easy answer and no off the shelf solution for success. Here is a list of guidelines that should get you started and help you in your quest.

To make the long list easier to read I will split it in three (3) pages:



### *Set yourself up for success*

**1. Define early on what success means for you, the team and the business**

You have to have a clear idea of what you want. You also have to understand what team members and the management want. You also have to be aware that what people really want, what they say the want and sometimes even what they think they want are very different things. Try to be very honest at least with yourself. Success has different definitions for different people. If there is a big disconnect between these definitions you have a problem before you start.

**2. Believe in the project: idea, architecture, time, team**

You cannot have any kind of success if you are convinced you lead a team of morons to implement a stupid idea using the wrong architecture in a ridiculously short time. You have to really believe in the project to have a chance to success. This does not mean lie to yourself. It means do whatever you can to understand your concerns and work on them with the management. As for the architecture, it is best if you have a heavy word or if you are the architect.

**3. Understand the domain, the business requirements and the technical challenges**

You should be an expert in the technologies used for implementation. You also have to become very knowledgeable in the problem domain and the business case. This will help you understand the business decisions dropped on your head from upstairs and also will help you stand a chance at negotiating them.

**4. Know your team: strengths, weaknesses, ambitions and personalities**

Software is created by people. Your job as a "tech lead" is to support them in doing that, both from a technical point of view and at a human level. You want to lead a team of motivated and enthusiastic people. But each person gets motivated by different things.

**5. Have a plan as a result of a planning activity**

"Plans are useless but planning is essential" – (Dwight D Eisenhower, US President, general 1890-1969). Planning will make you think about the problems you face in detail. Also keep in mind that "a plan is just a list of things that ain't gonna happen" – (Benicio Del Torro in "The Way of the Gun").

**6. Be part in the design of everything**

This does not mean do the whole design. You want to empower team members. But your job is to understand and influence each significant subsystem in order to maintain architectural integrity.

**7. Get your hands dirty and code**

Yes you should take parts of the code and implement them. Even the least glamorous parts. This will help you not getting stuck alone between management and the team. It will also help you gain respect in the team.

**8. Act as a communication proxy for your team**

In long complex projects with big teams communication is one of the most complicated aspects. The more people you have involved in solving a problem the bigger the communication matrix becomes. Since people need information to be able to make the right decisions this will lead to an exponential increase in the time consumed for communication. Agile methodologies alleviate this problem. But in the end it is up to you to propagate important information to the right people.

**9. Make sure everybody understands the big picture: their work has implications**

This will help you greatly because will allow team members to design and implement in a way that you don't have to fight. It is also hard work from your part.

**10. Fight for architecture and design consistency**

Doing the right thing from the design and architecture point of view is not more costly. It is actually cheaper in every project longer than a couple of months. Every early investment in architecture pays for itself later during integration and maintenance. Even if you have to admit an occasional hack or prototype in the code base you should contain it in very specific modules.

**11. Know the status of everybody's work and detect slippage**

This allows for corrective actions and for early communication with the management. You don't want to be caught by surprise. Remember that during 90% of the allocated time for a task the code is 90% complete.

**12. Record technical debt if you need shortcuts but try to maintain architectural integrity; report the debt**

This one is very important for products that will have multiple releases. Technical debt should be analyzed at the beginning of each iteration.

**13. Use the process that makes sense in your particular case**

Tough one. Sometimes (most of the times?) the process is not up to you. In the enterprise usually the process is pre-decided. But always keep in mind that the process in itself means nothing. It is the people who give meaning to the process. Good people can make the worst process work while the wrong team cannot make

any process work. Waterfall can be implemented in a very agile way and the agile methodologies can be applied with "*rigor mortis*" agility (see **The Agile 800 Pounds Gorilla**).

**14. Avoid dogmas – question why everything is done the way is done; make sure everybody else knows the reasons**

Sometimes I hear from programmers: we are agile and combine XP and Scrum and we also do TDD (Test Driven Development – I still hope for a TDD that means Thought Driven Development). The questions that pop up in my mind are: Do you need all those? Do you "really" do them by the book?

Anyway the point here is don't do anything just because it is the way it has always been done. Understand why. Then explain the reasons to all team members. Rinse and repeat.

**15. Avoid design by committee; listen to everybody but make your own decisions**

No good design is born from referendum. There are lots of people making wild exotic suggestions when their a$$ is not on the line. There are also excessively prudent ideas born from fear. Even with good ideas you have to filter them and make them yours before you can include them in the design. A good architecture and a good design is usually born in one mind, an open mind that looks around. The obvious example is Linux.

**Next page: Build your relationship with the team**

***Build your relationship with the team***

**16. Gain the team's respect with the quality of your work and by doing what you are preaching**

If you want something from a group of people you have to be an example for that something. Show them quality if you want quality. Show them enthusiasm if you want enthusiasm. And so on… but with a team of programmers don't forget to show them good code and good design.

**17. Be fair**

Being perceived as fair by your team is essential and unfortunately this is really hard to get because you have to say no sometimes. No, to good ideas that will improve the product. No, to bad ideas that will improve somebody's resume. The way you say "no" and when you say "no" and to whom you say it makes all the difference.

Also be fair in allocating work. If somebody is more capable and is able to do more, be sure to compensate that person in some form. A programmer whom I respect deeply once said to me: "I finished my work and then I started to help John. He was way behind and the boss asked me to give him a hand. After a week we were still behind because John's code was a mess and I had to basically start over. And at that time, in a meeting, the boss screamed at me because I didn't do John's work faster!". The story speaks for itself.

**18. Admit your mistakes**

Goes hand in hand with the previous advice. It takes courage to admit your mistakes and to make yourself vulnerable in front of your team, but this will build trust between you and the team. Just be careful not to make mistakes too often.

**19. Publicly recognize both team's and individual members' merits**

Think about a scenario: an emergency arises and you ask a team member to take care of it. The person solves the crisis, perhaps with a lot of personal effort. Then,

in a meeting with the whole team, and higher management, you say "We worked hard as a team and we solved it". If you don't mention the person by name they will never forgive you.

**20. Don't blame anybody publicly for anything**

In fact as a tech lead you cannot blame anybody but yourself for anything. The moment you blame a team member in public is the moment when the team starts to die. Internal problems have to be solved internally and preferably privately.

**21. Build morale and confidence by offering early victories to the team and to its individual members**

I cannot over stress the importance of this advice. Victories bring victories and sometimes you have to set up the first one. If you offer the chance to early victories the team will gel faster and the enthusiasm will create other successes. The best scenario is when somebody manages to accomplish something he was sure he cannot do.

**22. Match people and tasks based on skills and their personal preference if possible; explain your decisions**

Another difficult skill to master. Everybody in the team wants the glamorous parts that use the latest technology. You have to match every task with a person and you have to make them happy with what they got. It has to be challenging and comfortable at the same time for all of them. This is no easy feat but it can be done. You have to go back and look at what motivates each person in your team.

**23. Work the estimates with the team don't come up with them**

If you don't do it you will be perceived as not fair. Also keep in mind the estimate, to have any meaning, has to be done by the person who will do the work.

**24. Mentor people**

It is your job to raise the education level of your team. By doing this you can build relationships at a more personal level and this will gel the team faster and harder. It is very effective with more junior people in the team but there are ways to approach even more senior members, just try not to behave like a teacher.

**25. Listen to and learn from people**

Even if you are the most experienced developer on the team you can always find new things and learn form others. Nobody is a specialist in everything and some of your team members can be domain specialists who can teach you a lot.

**26. Explain your technical decisions**

There are many reasons for this advice. By trying to explain your decision you get to understand it better. You get valuable feedback. You ideas are bought faster. You make people understand that you value their opinion.

**Next page:** [Build your relationship with the management and business people]
**Previous page:** [Set yourself up for success]

***Build your relationship with the management and business people***

**27. Be sure you have authority along with responsibility**

Depending on the situation this might be either implied or impossible to get. Usually it's in between – you are backed up and given some authority but you have to work and get the rest as earned authority by making the team respect you. It is always useful to understand where you start.

**28. Be sure you get requirements and not architecture/design masked as requirements**

Sometimes business people fancy themselves architects, sometimes they just speak in examples. They can ask for technology XYZ to be used when what they really mean is they want some degree of scalability. Be sure to avoid hurting feelings but be firm and re-word everything that seems like implied architecture. Get real requirements. To be able to do this you have to understand the business.

**29. Explain technical decisions in business terms**

Don't explain your technical decisions to your managers the same way you explain them to your team. The business benefit derived from our decisions is all that matters here. Technical benefits, while sometimes understood, may look like over engineering.

**30. Try to be accurate in your estimates; avoid being too optimistic and don't push it with hidden padding; explain the need for padding**

Your managers are not born yesterday. They understand software development better than you imagine but they look at it from a different direction. Present your team's estimate and clearly add the padding as a result of unknowns in the presented estimate.

**31. Set reasonable expectations**

Don't be too optimistic. If this is your first project as "tech lead" be cautious when you predict time to market, quality and feature coverage. It is always better to promise less and deliver more than the other way around. There are always hidden dangers in any quest.

**32. Understand the relationships and dependencies with other teams or projects**

If the project is part of a bigger one you have to know who depends on you and what they want and who you depend on and tell them what you want.

**33. Accurately report the status with alarms, explanations and solutions; report any technical debt**

While being punished as the bearer of bad news is a valid concern, a good manager will always appreciate early warnings. Just be sure to bring at least a solution with your problem. The technical debt and its effects have to be communicated when they are significant since they can influence the business.

**34. Resist pressure for change in requirements, and more important for shortcuts...**

…but don't forget the requests might have a sound business reason. Be flexible, trade and negotiate if necessary.

**35. Be aware of politics**

I am not going to get in juicy details here since this was discussed extensively many times. Just be aware politics exist and the fact is natural in any human society.

**36. React to surprises with calm and with documented answers**

Never get carried away with refuses or promises when confronted with surprises. Ask for time to think and document/justify your answers. It will make you look better and it will get you out of ugly situations.

I got to the end of my long list and I realize it is still too short. But as always experience will be the best teacher. This is just a starting point. Good luck!