# JJ’s Reference Architecture

*Author: Jan-Joost van Zon*
*Date: December 2014 – July 2017*
***[Under Construction]***

# Team Management

## *Contents*

## *Introduction*

This documentation section describes techniques you can employ for the hands-on technical leading of a software development team.

## *Focus Points*

It is still debateable, whether everything falls into these key areas. I am trying to structure this…

- Dialog
- Support
- Direction
- Planning
- Strategy

- Keep your cool
- Be open to other views

## *Dialog*

- Is the one I might forget about, but if I put extra focus on it, it would help a lot.
- Exchange ideas between team lead and team member.
- Exchange ideas between team members.
- Get to know eachother to make optimal 'use' of skills but also to enhance morale en good feeling, mutual understanding and promote openness.
- Assess what the team member's problems are, regarding the way he feels about the work.
- Dialog should sometimes be limited too, because this job requires focus and quiet often, in order to do it well.

- Periodic workshops mostly from team lead, but could be from someone else, to present an idea.

### Periodic Individual Meetings

- Periodically talk to each developer individually about work done. Ask:
- Currently working on
- Done this period
- Problems encountered
- Large time-consumers
- Positive notes
- Other points of attention
- Solutions to the problems
- Next work to do
- Compliment on good work
- Good question: 'What way of working, would make you enthusiastic? Not looking at the way we work here.'
- Good question: 'Can you tell me some interesting new things you found out?'
- The team lead also asks himself the same questions.

### Reporting to Management

- Lead makes a report of issues discussed, preferably not grouped by employee, but grouped by development area. Employees can be mentioned with tasks.
- Lead discusses report with management.

## *Limiting Dialog*

- Prevent people from getting distracted by too much dialog.
- Promote sympathy for people that have ear plugs with music on.
- Only bother people when useful.
- Many people need to focus to program.
- You should be considerate towards people that need undistracted focus, ask more carefully if they can be disturbed and make them know, that is OK to say no to that.
- There are people that seem to handle distractions better, although I doubt their productivity does not get harmed.

- But perhaps I am wrong and these are the people that you can bother more often to vent about am issue.
- Consider moving a conversation to another room, so others can focus.
- Not speaking too loudly also helps a lot.

## Support

Support from the team lead is mostly inward directed toward the team members. But support is also has an outward direction from the team towards the rest of the organization or to the customer.

- A strategy of giving team members more freedom.
- Give people the responsibility, that their code works well and that others can still read it and adapt it.
- Support with solutions and techniques.
- Give examples
- Lead by example.
- Formulate suggestions with prefixes like "I would do it like this." or "In my experience this works well."
- So do not talk in definites, allowing the other person to form their own opinion and understand why or why not to take a certain route.
- You are going to have to accept that things will be programmed in a less than perfect way, or not precisely the way you would do it, and allow programmers to grow into making things better on their own.
- Look for a compliment to go along with it.
- When someone does not asks enough questions, as a lead you should inquire yourself.
- You can assume a relationship of equality, and still be a team lead.
- Do not think someone is stupid if that person does not know something
- Accept that someone else knows a thing or two you can learn from him or her.
- Do not be afraid to admit your weaker points, and work together to come to a solution.

### Question Rotation

- Questions from outside the development department are handled on a rotating basis.
- Everyone takes turns in a day of being the one to ask questions.
- This will make sure most of the time developers can continue undisturbed with their programming work.

## Support & Direction

- Support and Direction are very closely related.
- Support is more oriented towards the coworker: what can you do to make him do a better job and feel better doing it and advance in his job.
- Direction is more about how can I make the work be beneficial to the client or rest of the organization. It is more outward directed.
- Both are about how can we do the work optimally to serve the goals.

### Developer Skills

You can test someone's skills in something by giving them a specific task, that requires one skill in particular.

Meta-skills:

- Research
- Documentation
- Bug fixing
- Process flow programming
- API programming
- UI design (not graphical, just the screen layout and program flow)
- Programming for reusability
- The worst-practices and almost worst practices lists from the architecture documentation.

Tools & techniques:

- Database development
- SQL

<TODO: Add more… Could be a pretty extensive list.>

## *Direction*

### Miscellaneous Pointers

- Guard the main outlines
- Accept that people work outside the borders.
- Be involved at the beginning of a task at the end of a task and during the execution of a task too.
- If your involvement starts in the middle, do not try to to stear it too much into another direction.
- You could find the biggest problem area and then suggest that as a point of improvement.
- Meetings: (borders the topic of Dialog)
    - Try keep focus in meetings, so things do not go off topic too much.
    - On one hand I dislike it, when a meeting about a topic that could have been 30 minutes turns into 5 hours of a whole range of topics with only loose ends and inconclusives at the end.
    - On the other hand I dislike it, when the response to a proposal is: that is not what this meeting is about.
    - The intermediate version would be brief attention to off-topics and main focus on the topic at hand.
    - It also depends on whether it is a general meeting or a conversation about a specific topic.
- API choice: Medior developers and juniors are *not* free to choose any API without collaborating with seniors, leads and architects. It just has too high an impact. Not that we should not be open to new API's.
- Framework development: Same point as API choice: juniors and mediors only adapt framework in colaboration with seniors, leads and architects.

### Behavior to Expect

There are a lot of undesirable situations you can be confronted with as a team lead. While others would not dare do anything, you are expected to do something about it.

- After someone agreed to do something (a certain way) 3 times, the person still does not do it (that way).
- Someone says he or she understood something, but when you look at the work, obviously that person did not understand.
- Seeing criticism as an attack.
- Someone saying a job is done, while not even 50% of it is done.
- Someone claiming to be an expert while knowing he is not.

Or less annoying (because you cannot really blame anyone):

- Someone thinks he or she is an expert, but the truth is he is not.
- Someone spends a week on something, while asking for advice, could have made that a couple of hours.
- Someone extremely talented will keep beating him or herself up over tiny details or just will not assert his opinion or be scared of other peoples' criticism.

### Vary a Developer's Tasks

What I often see, is that each developer is assigned to his application and never changes focus, because everyone feels, that this person is the 'go-to guy' for this topic. The motivation behind this is that it would be more economical to let the guy that knows most about it, do the work. That makes sense intuitively, but what's intuitive, isn't always what's right. It is an over-simplified view on the issue that will cost you money. You lose something when you do not vary a programmer's work.

- There is the risk that comes a developer becomes ill or falling away for another reason.
- It excludes other developer's ideas from the subject.
- It will limit the developer's personal development, because he never gets to do something else.
- It will encourage developers to not think about: "What if someone else had to work with my code?"
- The lack of critical peer-review could make the code quirky and hard to maintain.
- There is no opportunity for developers to learn from eachother.
- There is less incentive to review eachother's code, which can translate to bugs and increased development time.
- It will be more difficult to scale up the amount of developers on a job.
- It will be more difficult to move a developer from one project to another, for instance when there is not much work in one area anymore, or higher priority work in another area.
- It does not promote a joint way of working, that makes it easier for others to pick up eachother's work.

I cannot stress enough that it is a bad idea to let a only one developer work on a specific thing all the time. Did I mention that the the software developer's personal development is slowed down? I have never been in a situation where this costs more and I have seen productivity go up doing it.

Here are suggestions of how to spread and vary the work over multiple team members. Variations can be mixed and mingled at will. That is the point. Be flexible.

- Variation 1:

    - Junior front-end
    - Medior back-end
    - Senior framework
- Variation 2:
    - Senior coaches medior
    - Medior coaches junior
- Variation 3:
    - Senior reviews code of mediors
    - Medior reviews code of juniors
- Variation 4:
    - 1 developer does rework of previous iteration
    - Another developer does new features.
    - Every now and then they switch
    - That way rework is not in the way of new developments.
- Variation 5:
    - One developer answers questions coming from outside the department and each day a different developer does it.
    - If needed other tedious task go to that developer.
    - That way the other days you know you can keep working on actual software development uninteruptedly and you at most only lose that one day that it is your turn.
- Variation 6:
    - Each developer or a set of developers works on a different part of the application or major feature.
    - Perhaps a junion/medior pair can work on a feature together.
- Variation:
    - A developer gets a similar task as before because he is now proficient at it.
- Variation 7:
    - A developer gets intentionally different tasks than before, to spread knowledge over the team and enriching the developer's knowledge.
- Variation 8:
    - Some developers work on high-risk experiments.
    - Other developers work on low-risk productive feature development.
    - That way you create room for doing experiments that you do not know the outcome of yet or do not even know how long it is going to take or if it gives any useable results, without it actually stalling productivity completely.
- Variation 9: (very specific)
    - Seniors do:
        - Task management
        - Code review
        - Architecture
        - Framework development
        - Data model design for the most part
        - Software design
        - Research / orientation of possible solutions
    - Mediors program:
        - Business logic
        - Data model implementation
        - *Some* framework and data model design but in tight collaboration with seniors.

- Juniors do:
    - Front-end development, but under supervision so that patterns are followed.
    - Repetitive tasks, but only ones the junior can handle.
    - This may sound a bit deprecating, but sometimes a junior developer, when asked to do something in a very specific way, apply it the same all over the place, do it totally differently when you turn your back. That says something about the need for guidance. You have to monitor junior developers.

### Upward Management

- Lead does ask critical questions to management about strategy and decisions.
- Lead advises management about (future) employees to hire.

### Budgeting

- Lead investigates software and hardware requirements and opportunities to use newer software, offer better quality, quantity, service or reduce costs.

<TODO: Write a little more about it.>

## *Planning*

Borders the topic of 'Direction'.

### Tasks

- Split up tasks into multiple distinct pieces of work.

  If the task is to broad, some people that think they can add another 200% of work to it, that they made up themselves. It will be unclear where a task starts and ends.

- As the lead, always give an expected estimated time it will take.

  This will make someone aware of expectations, and of how deep to go.

- Lead must control task subdivision
- Lead must be asked for new task and discuss how to solve
- Lead must be informed of completed task
- Lead must be asked in case of unclarities
- Lead communicate the following rules

    - Begin Task -> Go to lead
    - End Task -> Go to lead
    - No Task -> Go to lead
    - Questions/unclarities -> Go to lead.

  Fill in the lead's name and basically hang up the wall.

- Data data model extension task should mention:

  '+ Add the basics around this data model to all the architectural layers.' Discuss verbally which layers. Use the Layering Checklist from the Software Architecture document's Appendices to make a selection.

- A task should involve all the required work, but what is not asked should be left out

  But this comes with problems.
  Ideally a developer knows which things to involve in a task, and which things to not involve.
  But in practice this is often not the case.
  One developer may not know what a task implies, another developer will involve too many side-issues.
  If you need to mention everything in a task, this is too much of a burdon on the one administrating the tasks.
  <TODO: Brainstorm and reformulate later.>

## Strategy

- Strategy is broad. It takes anything you could do and molds it to fit your current goal.

<TODO: Write something about it.>

## Keep Your Cool, Be Open to Other Views

<TODO: Write some more about it.>
<TODO: Use this phrase? All this heated discussion about subjective things. Can't we just keep calm and discuss pros and cons and try to solve the problem together? Nothing is black and white in the choice of tools and techniques. It is always a trade off.>
<TODO: Use this phrase? Be less sarcastic.>
<TODO: Use this phrase? Someone with a way of working that does not promote maintainability, might not just be stupid.>