Infolyzer

.io INSIGHTS AND OPTIMIZATIONS

5 Signs that your development team is not reaching its full potential

AND WHAT YOU CAN DO ABOUT IT

www.infolyzer.io | fredrik@infolyzer.io

1 Long lead times

SYMPTOMS

- Misunderstandings are common. Requirements are unclear, and it is difficult to get more information
- Developers are unsure of what to build, and sometimes they are even working on the wrong things
- Testers report bugs that are already fixed, just because they weren't properly informed to which environment the code was deployed to
- The typical response to a problem is to call a meeting, which typically results in just another meeting
- Even simple features take an unreasonably long time to develop
- Everyone has limited bandwidth, especially key persons
- Code get stuck in pull requests and code reviews
- Everything requires excessive documentation or sign-offs from managers

CAUSES

- Requirements documentation is either non-existing, hard to understand, or scattered across multiple channels
- Unclear communication from the stakeholders, the product owner or the project manager
- Lack of understanding of (or respect for) how developers like to work
- A business environment where interruptions are common and it's hard to stay focused
- Business politics manifested as a lot of ceremonies and rigid processes
- Employees are assigned to multiple, concurrent projects, often ending up working overtime to get by, or not performing optimally in any of the projects
- Negativity in the team, resistance to change and a general lack of buy-in
- People of lower morale slacking off since no one notices, a.k.a. freeloaders

SOLUTIONS

- Make sure to convey the Why-perspective, and establish both a long-term vision and short-term missions
- Scrutinize every process and strive for continuous improvement. Involve everyone in this work; this is not something that always should be done by managers. Trust breeds creativity, let the team work out what's best for them
- Increase the transparency of the work. Are people productive or just being busy?
- Keep in mind that the transparency that comes from collaboration and peer pressure trumps rigid control mechanisms
- Minimize "catch-all" meetings and always have a clear agenda that is communicated in advance
- Allow developers to work undisturbed in larger blocks of time as often as possible, preferably whole days
- Are developers being micro-managed? Nothing destroys motivation as lack of trust. And unmotivated team members won't bother to go the extra mile
- Foster a culture where it is ok to fail and ask for help without losing face. This will increase quality over time since every failure presents a learning opportunity

2 Low quality

SYMPTOMS

- Buggy releases that are riddled with regressions and undesired side effects
- The system requires a lot of manual testing
- Testing is complex, hard to set up, or require tooling not available to everyone in the team
- Developers are reluctant to touch certain parts of the code since they don’t want to break anything by mistake
- Lack of trust in the existing test suite
- Inconclusive test results, i.e. some tests are allowed to fail
- A lot of contradictory, half-baked solutions in the code base

CAUSES

- You have a huge technical debt and it’s not being prioritized
- Insufficient communication between stakeholders, product owner and the team
- Lack of accountability
- No sense collective code ownership
- Insufficient automated test coverage of the code base
- Cowboy coders not following the agreed upon guidelines
- Poor handovers, non-existing onboarding
- Lack of standards, procedures, and checklists
- Developers not having enough time to finish refactorings they have started
- Long-lived feature branches in source control
- Bugs introduced by improperly resolved merge conflicts

SOLUTIONS

- Don’t neglect refactoring and make sure you keep the technical debt under control
- Enforce a reasonably strict set of coding guidelines, but keep in mind that too much can kill team spirit
- Prefer pair- and mob programming over traditional code reviews to spread good habits and increase knowledge sharing
- Write tests for the important parts of the system, but remember that high test coverage is not a goal in itself
- If possible, rewrite the parts of the code that “cannot be tested”
- Avoid half-baked solutions and architectures at all cost
- Always finish what you start. If you decide to change something, make sure you change it everywhere
- Have a clear source code branching strategy and make sure everyone on the team understands it

3 Fear of deploying to production

SYMPTOMS

- Every system release is regarded as high risk and involves a lot of (often uncalled for) anxiety
- The team gets relieved — and sometimes even surprised — when the release didn't break anything
- Releases are done outside office hours, even if it's not mandated or necessary
- The management demands on-call personnel 24/7 just in case, even if it's not warranted
- Code freeze periods and excessive manual testing before every release

CAUSES

- Uncertainty about what code actually ended up in the binaries and packages being released
- The release process is key-person dependent, and often there is only one person that knows it
- Every deployment involves many manual steps that are hard to perform without having the necessary competencies
- Trust issues; "We can't give everyone access to the production environment"
- Non-existing or outdated configuration management documentation

SOLUTIONS

- To avoid the code uncertainty: make sure that there is a clear branching strategy in place that everyone understands
- Automate as much of the deployment process as possible
- Involve more people in the process, everyone on the team should be able to perform a release
- Create checklists and keep the documentation up to date
- Revise your branching strategy to minimize the need for code freezes
- Strive for a process that allows for small, frequent releases instead of a big-bang-style release once a quarter

4 Firefighting culture

SYMPTOMS

- The team skips safeguards protocols such as Definition-of-Done to meet customer deadlines
- Customer expectations are always too high and deadlines too tight
- The daily agenda is constantly changing, leading to a lot of forced context switching and micromanaging
- There is never enough time to create high quality functionality, just barely working hacks
- Source control hell

CAUSES

- Stakeholders and salespeople make promises to customers without first clearing them with the team
- Focus on developing and improving features instead of fixing problems
- Underestimating development time and ignoring all “hidden” tasks in the estimation process
- Not dealing with the technical debt
- Having a lot of concurrent feature development for different customers quickly creates the need for long-lived branches in the version control system
- When the time comes to merge a feature branch, the original developer is working on something else, leaving the remaining developers left to their own devices

SOLUTIONS

- Shift to a more long-term focus. Realize that all these quick fixes inevitably will introduce bugs and general disorder in the code base. And that it will cost a lot more money further down the road, than it would if done right at the outset
- Track down the root cause to all errors instead of fixing the symptoms
- Consolidate and simplify the code base as you go along
- Get better at expectation management, both internally and externally
- Track hours to get better at estimating — and realize that estimates are, at best, qualified guesses
- Consider Kanban and WIP limits as a means to minimize concurrency and long wait states
- Realize that refactoring is a continuous, ever-ongoing process. Just like brushing your teeth for one hour straight once a week won’t be good enough, an occasional, focused refactoring effort will have very little impact on the overall quality of your project

5 Negative team dynamics

SYMPTOMS

- The team have a “We vs. Them” mentality
- Some team members are not pulling their weight or dodge responsibility
- Alliances take form: teams within the team, rockstar teams, A/B teams
- Developers ignoring prioritized tasks in favor of their own agenda
- Ostracism and exclusion
- Some team members get special treatment
- Abnormal amounts of sick leave

CAUSES

- Unresolved conflicts
- The management has failed to establish a set of clear boundaries
- Rockstar developers or key people with special benefits while not respecting the rules that apply to the rest of the team
- Prestige and egoism
- Persons with lead roles exploiting their positions for personal gain
- Micromanagement
- A corporate culture bias for written communication over talking face to face

SOLUTIONS

- Keep an ear to the ground and act immediately upon first sign of trouble; if there is smoke there is usually a fire
- Make sure that information intended for the entire team really reaches the entire team, and not just the ones you drink beer with after work
- Ensure that the agreed upon rules are followed by everyone — no exceptions
- Make an effort to hire motivated people that fit your company culture. Second, try to place them in a position where they can thrive
- Don’t delegate your HR-responsibilities to the informal leaders in your department
- Written communication is great, but keep in mind that it can often be perceived as harsher than actual face-to-face meetings
- Pull requests and written code reviews are great for documenting the discussion around a piece of code, but it can easily turn into a vortex of misunderstandings and annoyance. I prefer to use pair- and mob programming as much as possible, effectively making code reviews and pull requests a final, administrative step
- Make sure to regularly gauge the mood in the workplace. If you decide to interview the team, conduct the interviews individually. And send out the questions ahead of time. Many developers won’t speak their mind in a group setting, and everyone gives better answers if they get a chance to think the questions through in advance