Do less testing

Disclaimer

Business

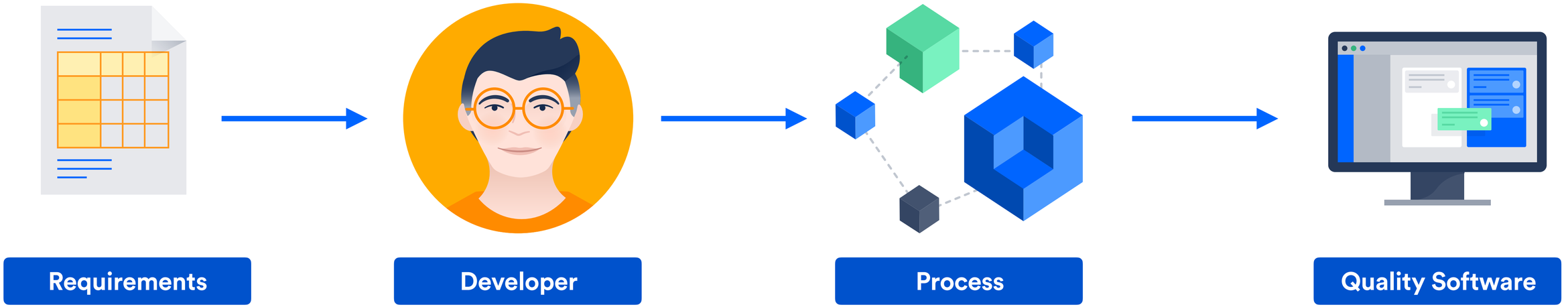
Testing ≠ Quality

“

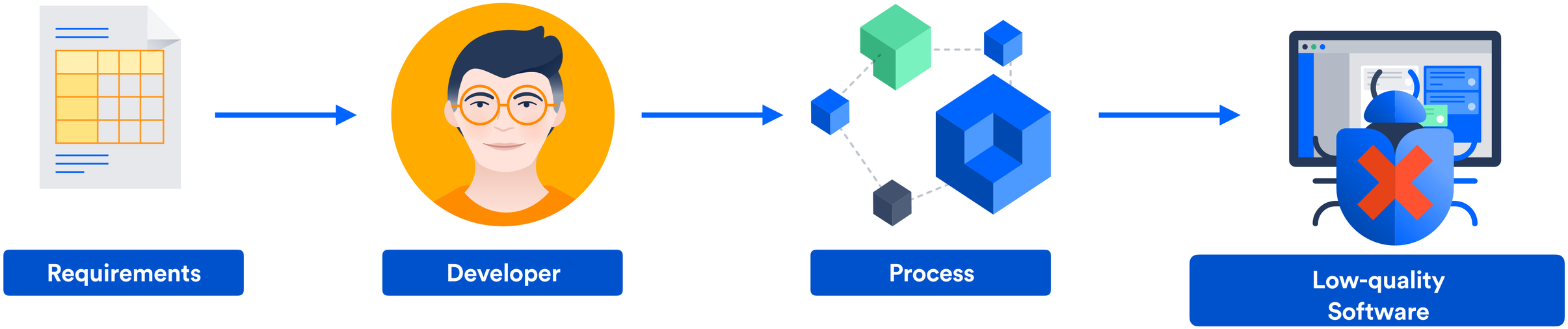
Effective software teams are all alike; every dysfunctional team is dysfunctional in its own way

L. TOLSTOY

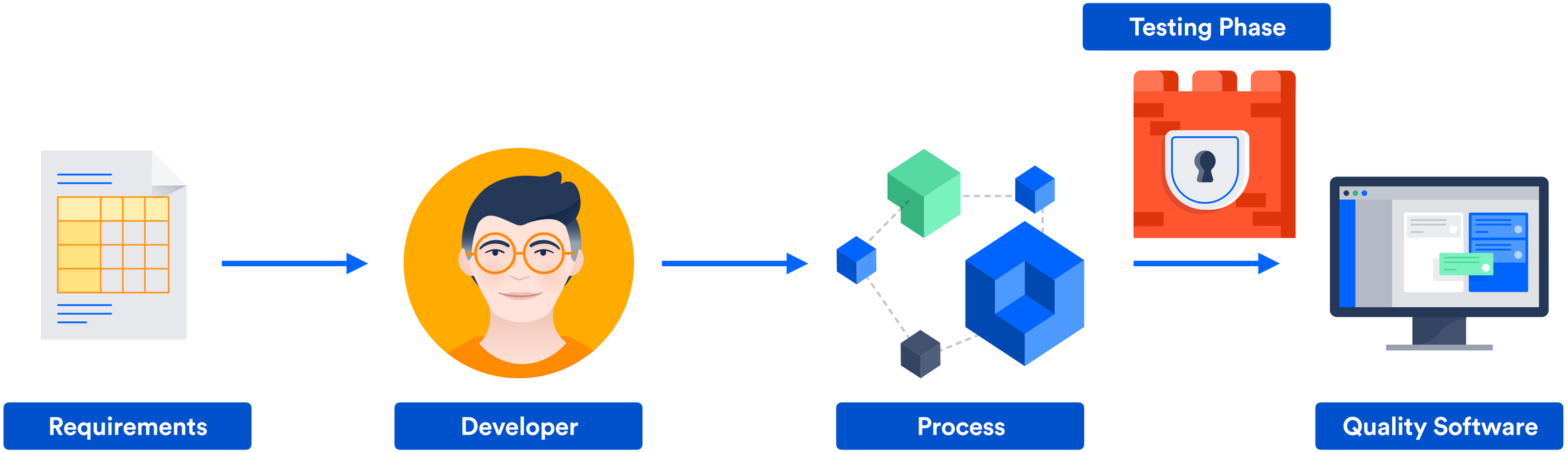
Effective software team



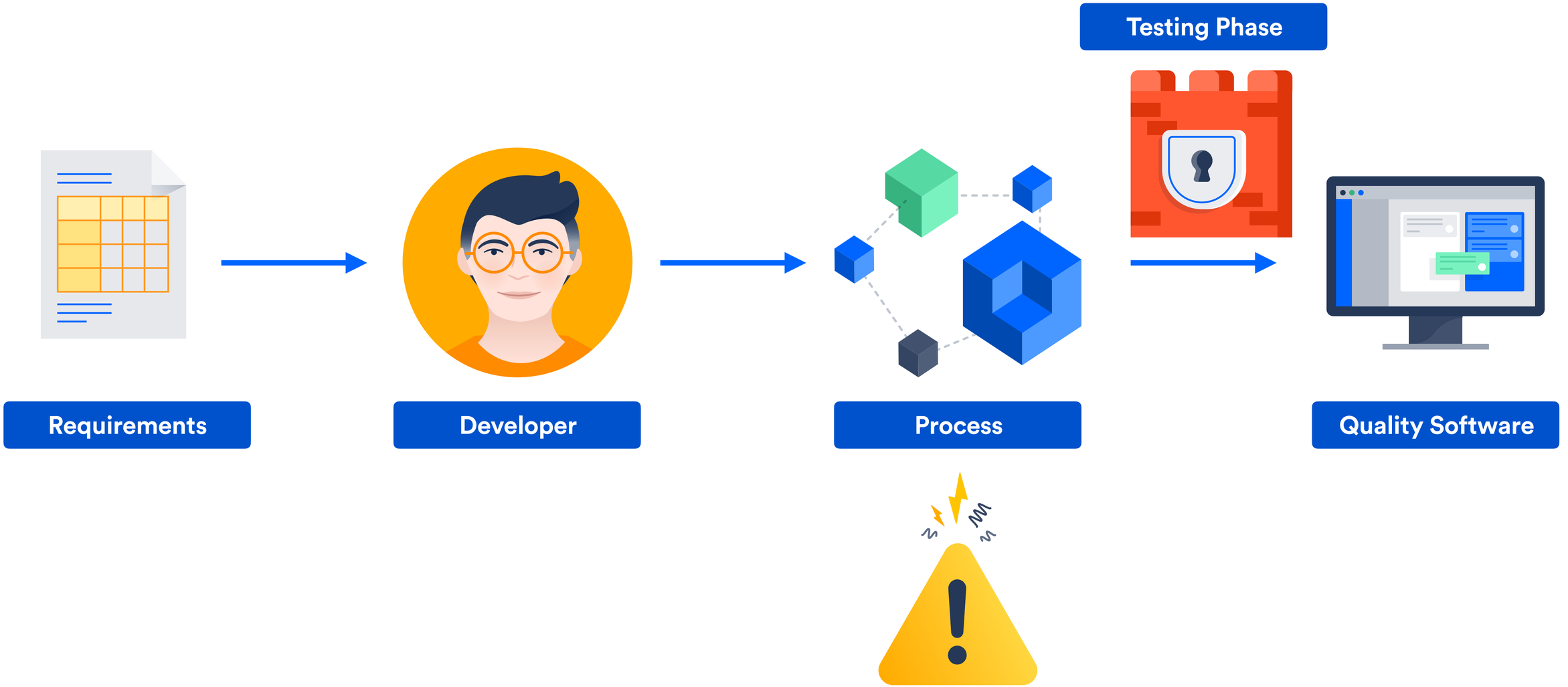
Dysfunctional software team



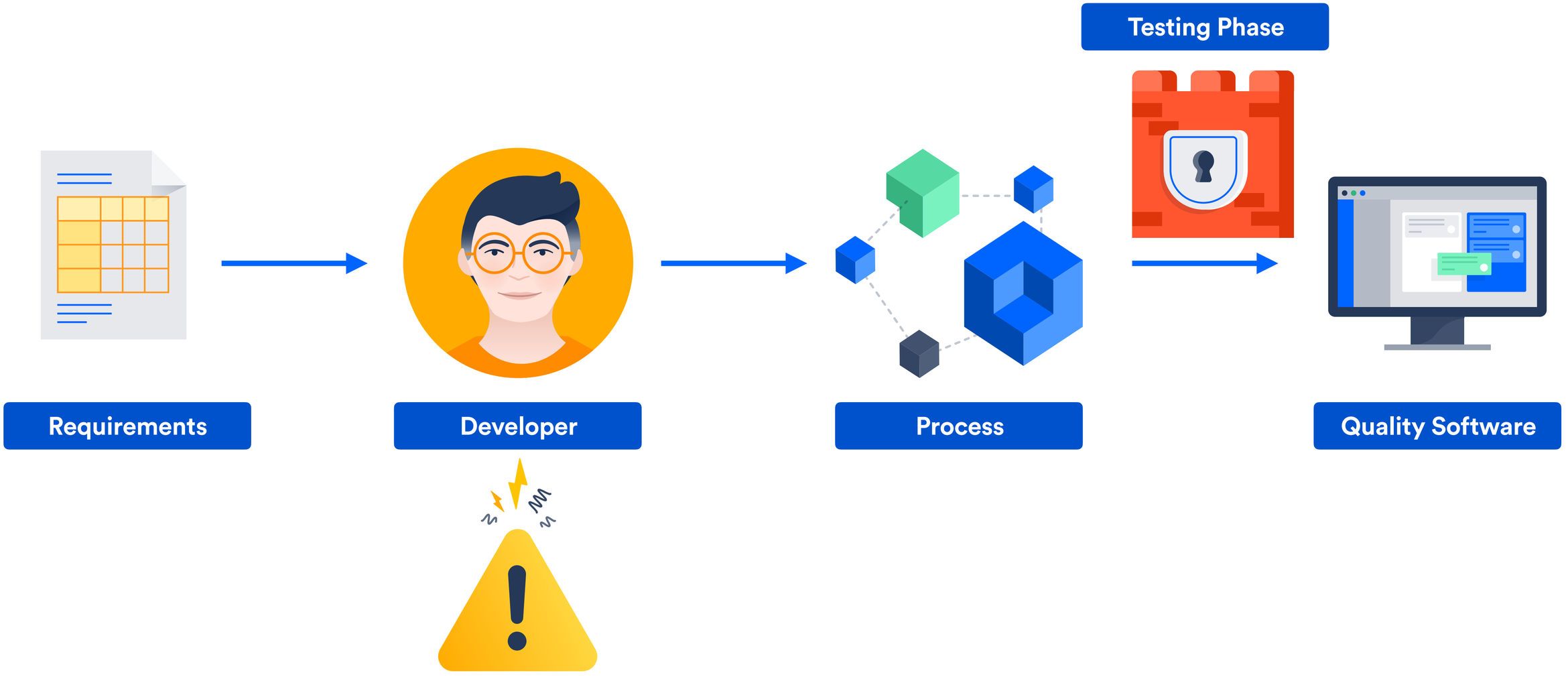
Dysfunctional software team



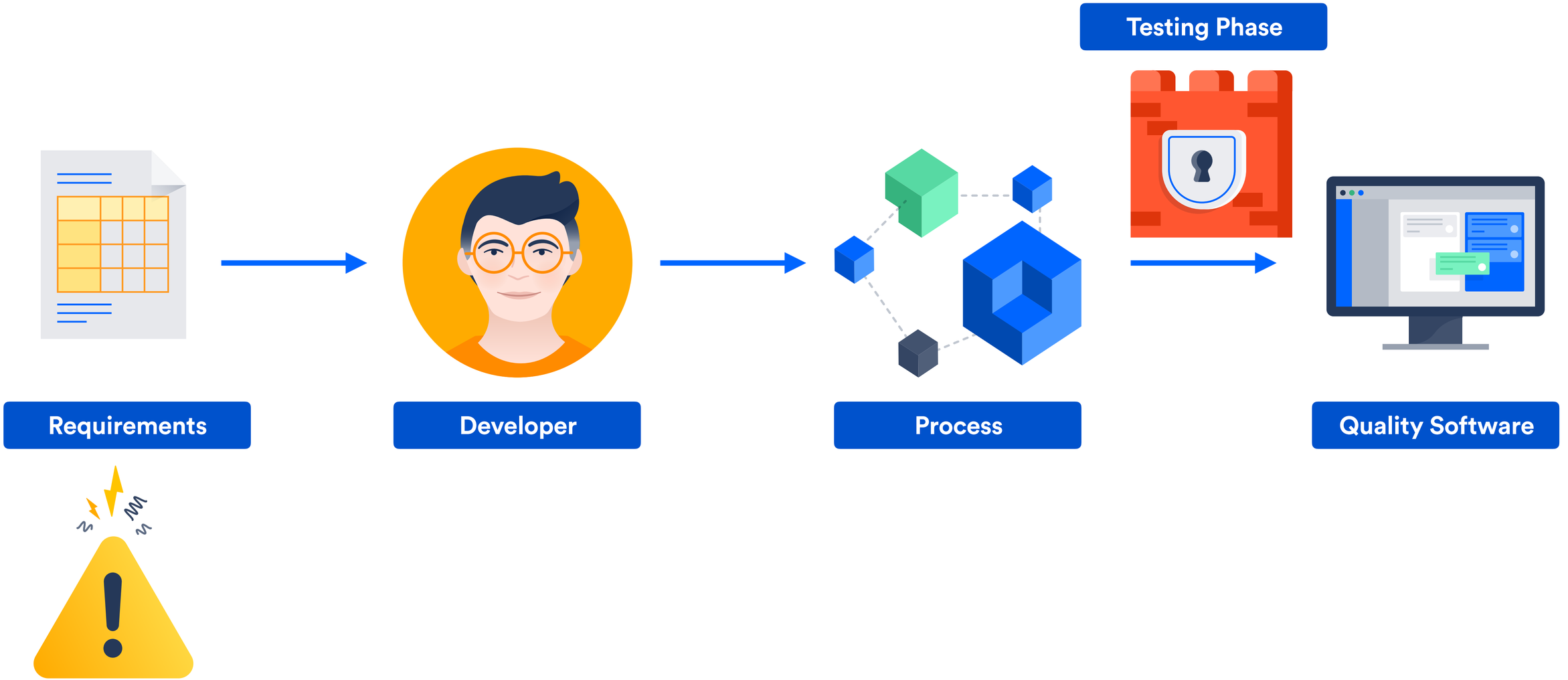
Dysfunctional software team



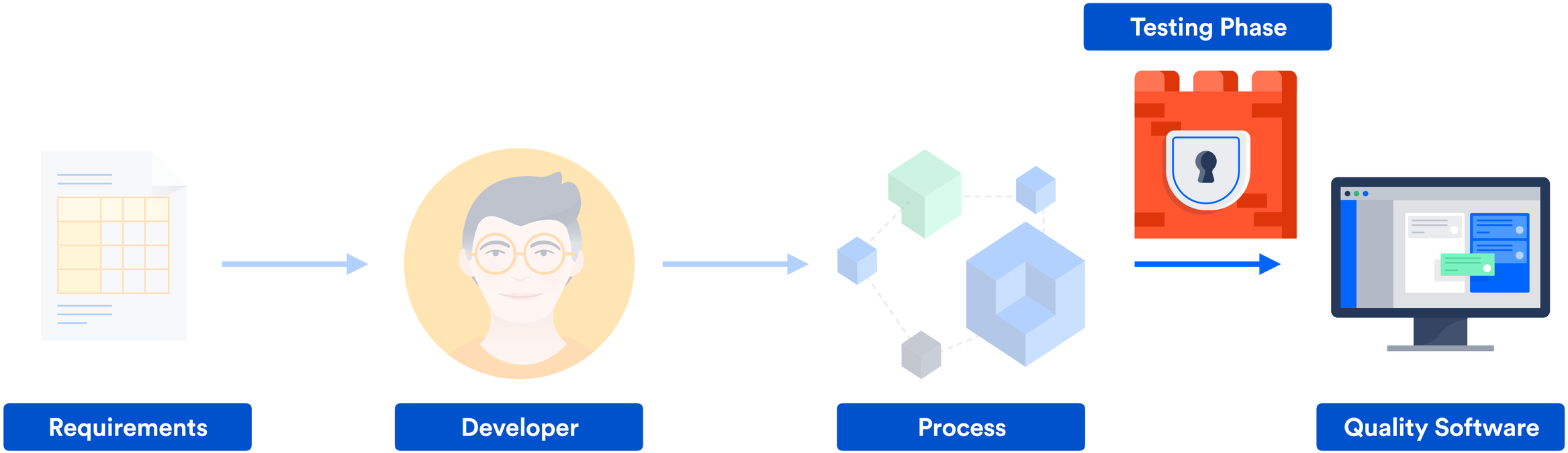
Dysfunctional software team



Dysfunctional software team



Dysfunctional software team



Conflict in roles

Developer

Writes the functional code

Tester

Writes the test code

Conflict in aims

Developer

Wants to release

Tester

Wants to block release

Conflict in attitude

Developer

Does not care about quality

Tester

Cares about quality

Conflict in perception

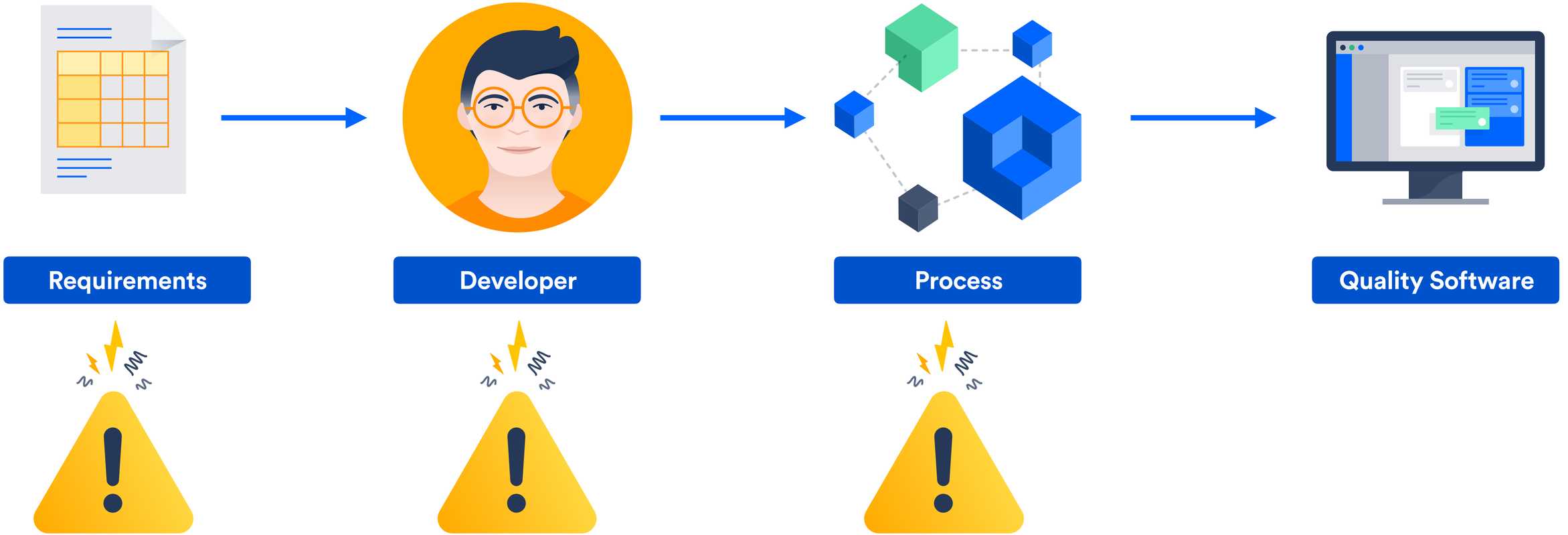
Developer

Delivers explicit value to the business

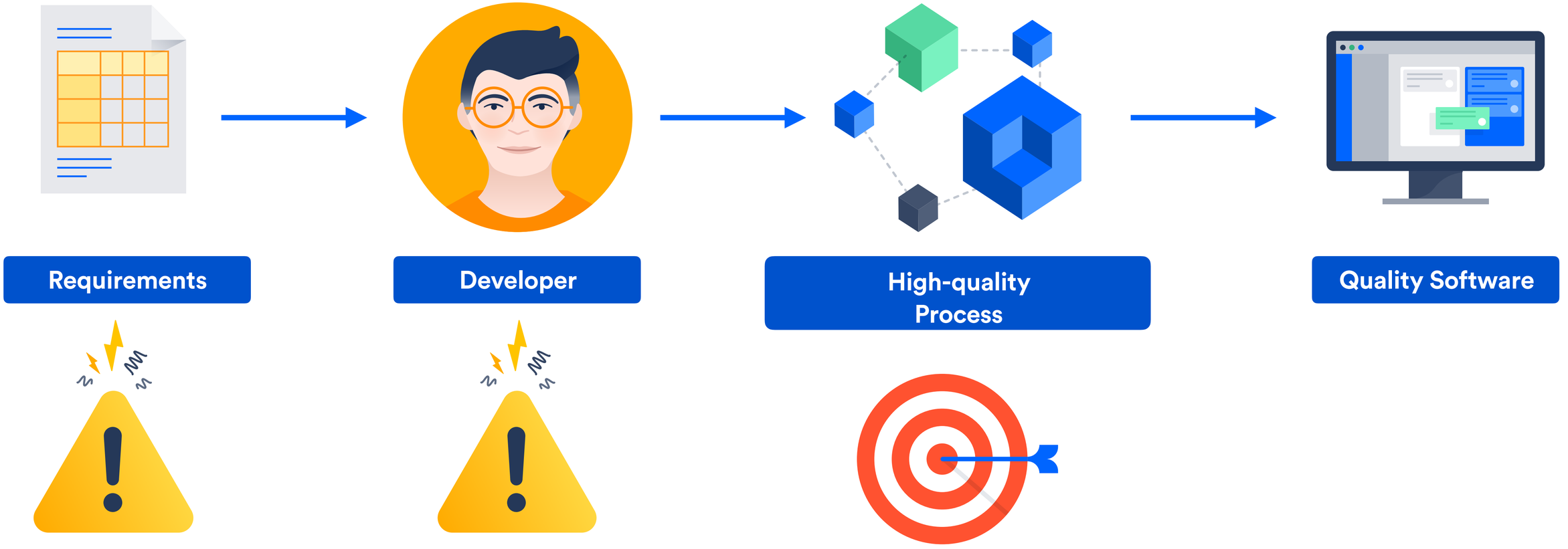
Tester

Value is implicit and unmeasurable

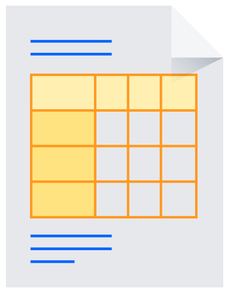
Dysfunctional software team



Dysfunctional software team

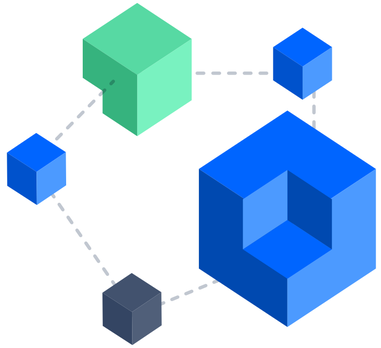


Dysfunctional software team



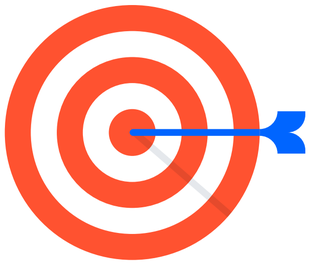
Requirements

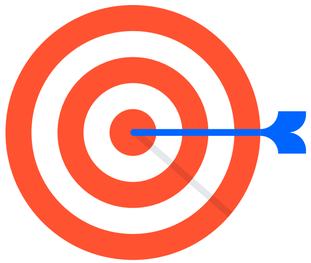
High-quality Developer



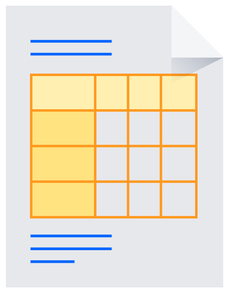
High-quality Process

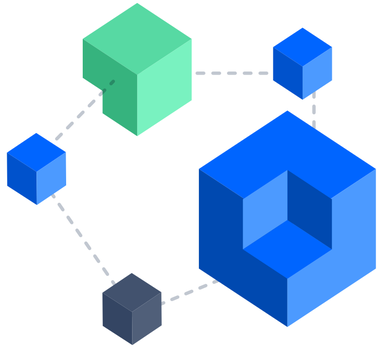
Quality Software





Effective software team



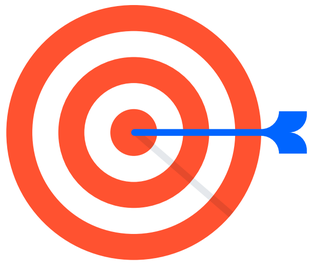


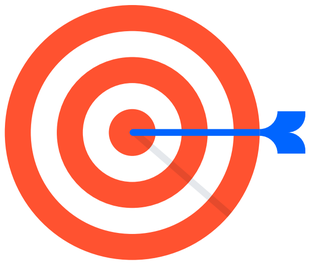
High-quality Requirements

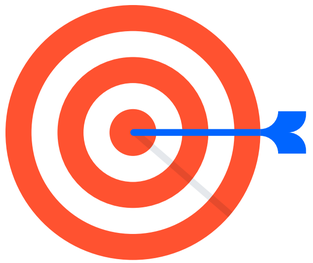
High-quality Developer

High-quality Process

Quality Software







Testing ≠ Quality

Do less testing

What is testing for?
What are the ongoing costs?

Manual testing

Can we find bugs?

Automated testing

Is it safe to release?

Automated testing is not cost-free

Framework & implementation

Ongoing maintenance

(Virtual) Hardware

Time to run

Automated testing is not cost-free

Framework & implementation

Ongoing maintenance

(Virtual) Hardware

Time to run

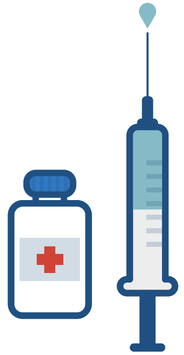
JOURNEY TO CONTINUOUS DEPLOYMENT

Reducing time between releases from:	Acceptable testing:
Months to weeks	Several days of manual testing to find bugs
Weeks to days	
Days to hours	
Every commit	Absolute reliance on fully automated pipeline

How to do less testing?

How to spend less time doing testing?

Tactics to reduce/replace testing



Prevent

Identify potential problems before coding

Mitigate

Reduce the impact of problems to end users

Listen

Adjust the quality bar based on your users' actual experience

Examples

Prevent

- Kickoffs to identify edge cases
- Paired development
- Train your developers in quality thinking

Mitigate

- Monitoring
- Staged deployments with automatic rollbacks
- Feature flags

Listen

- Realistic and consistent bug policy
- Impact-driven priorities
- Don’t sweat the small stuff

Takeaways

Mindset
Goal is quality software, not testing activity

Trust
Developers are allies with the same goals

Automate
Machines tell you that deployment is safe

Measure
Testing has costs, especially time-related

Replace
Alternative ways to engineer quality software