Distributed Software Development

How to Build Your Product and Create Your Team: ensuring the availability of developers' resources for your business.

White Paper

intetics
Where software concepts come alive™

Table of Content

Distributed Software Development Introduction

Nowadays, distributed computing – a model in which an application's components are shared across multiple computers – has become integrated into our work and private lives alike. Anybody who googles a question is using distributed computing. Distributed system architectures provide businesses with countless services, strengthened by deep processing and computing power. And distributed software development is just one of the many things that this kind of computing makes possible.

Distributed software engineering enables IT teams to collaborate on applications and software across geographical lines. Modern tools and collaborative techniques have made distributed development work more effective in recent years.

In our White Paper, we will look at what exactly distributed software development is, effective tools and techniques, main applications, and best practices.

The Full Potential of Distributed Teams for Your Business

A distributed team, in its simplest definition, refers to a group of individuals that work together while residing in different geographic locations. They may use instant messaging, email, videos, and voice conferencing services to facilitate smooth communication and collaboration. Such groups or teams can also work together asynchronously and across organizational levels. How are distributed teams even possible? Fiber optic technology has significantly enhanced the possible level of off-site communication. Now, companies can hire the best talent without having to worry about geographical restrictions causing an impediment.

Covid-19 boosted the switch to distributed teams, as teams could no longer meet in person and were required to work from home. But even as vaccination rates are rising and the world is beginning to return to a "new normal," there are indicators that distributed teams and remote work is here to stay. For instance:

- 99% of employees that are currently remote would prefer to stay remote for at least some of their career.
- 56% of the US workforce have job duties that are partially or fully compatible with remote work.

And the full potential of distributed teams hasn't even been tapped yet; the model will continue to evolve as new technologies emerge and with new methods of internet connection. Take, for example, Elon Musk's Starlink. This service aims to make web access and global communication more accessible for people in remote regions of the world – opening up new geographical areas to work that may not have been possible before.

In its current state, distributed software development offers a wide array of benefits when multiple teams are working on the same application, including:

- **Time Effectiveness**

  A company that uses distributed software development can have employees located all over the world, enabling its teams to work in various time zones. When developers are scattered strategically, the organization ensures that at least one team is always available. As such, companies that require 24/7 support can heavily benefit from distributed development.

- **Cost-Savings**

  By hiring quality IT staff from other countries, especially in the Eastern European area, companies save on salaries, operating fees, and real estate rental costs. According to Global Workplace Analytics, for each remote worker employed, the average organization saves over $10,000 per year on rent alone.

- **Talent Pool Diversification**

  With a distributed development team model, companies can erase territorial boundaries and improve their flexibility. Businesses get access to a virtually unlimited talent pool of experienced, skilled developers residing in all time zones.

Summed up, the three most important benefits of a distributed software development team are money, people, and work output.

Preconceptions Regarding Distributed Development Teams

THERE ARE SOME PRECONCEPTIONS ABOUT DISTRIBUTED SOFTWARE DEVELOPMENT TEAMS; WE'LL TAKE A LOOK AT SOME OF THE MOST COMMON ONES AND FIGURE OUT WHAT'S FACT AND WHAT'S FICTION.

Preconception 1: Poor Quality of Work

When many people picture a distributed development team, they imagine a crew of underpaid, part-time developers that are overloaded with projects and slipping on quality. However, this is certainly not the case. It's true that shoddy teams were more commonplace about 15 years ago – but, nowadays, the distributed software development market is exploding, and, subsequently, dev companies need to work HARD to rise above the competition.

Thus, outsourcing vendors meticulously vet potential employees, ensuring that their clients are only provided with reliable talent.

Preconception 2:
Distributed Development Teams Can't Be Adequately Managed

Although it can be challenging to implement Agile methodology within distributed development teams, it is still doable – as long as you adjust your operations and distribute workload evenly across all teams. You need to ensure that each member has an approximately equal workload and that all employees understand their roles. There are also many practical online tools at your disposal, which can help you improve management and planning processes. For instance, Jira, Azure DevOps, OmniPlan, Trello, and Monday are all great for organization, while Google Drive, GitHub and Dropbox work excellently for collaboration.

You should also keep in mind that you can hire a distributed team that has a project manager – in such a case, you would control the process at a higher level, and the project manager would take care of local management processes.

Preconception 3:
Distributed Development Teams Can't Communicate

Again, this is a preconception that has roots in truth but became outdated upon the advent of online collaboration and communication tools like Zoom, Microsoft Teams, Slack, and messengers that became platforms for communication. Nowadays, distributed development teams have access to a wide variety of tools that enable smooth, streamlined communication that is often advantageous over on-site meetups.

Real Challenges

While the aforementioned preconceptions are easily addressed with modern tools, this isn't to say that distributed teams have no pain points. There are a few key issues that need to be addressed when creating and leading a distributed development team: building trust, cultural/language barriers, technical alignments, and project/process management.

1

Building Trust

Trust is an essential element of a hyper-productive team; without it, there will be no cohesion between team members. Yet, when team members are distributed across various environments, time zones, and cultures, it's difficult to gain trust. If a team doesn't have a minimum trust level, it's easy to blame and criticize other groups when challenges appear. But when trust is strong, your distributed team members will create innovative solutions to work through difficult issues.

2

Cultural/Language Barriers

It is necessary to impose overlapping office hours so remote team members can bridge time zones and communicate, resolve problems, and collaborate on mission-critical tasks. By ensuring more overlapping working hours, your team will experience better engagement and interwork. Another consideration in this area: when your team members come from various cultural backgrounds, it's easy for misunderstandings to arise. Furthermore, team members from various regions might have various degrees of technology and skill expertise – which could lead to a hierarchy between your remote teams.

3

Quiet Participants

Quiet, unmotivated employees can be found in distributed and co-located teams alike – but it's a lot easier for them to be motivated when they can see their colleagues face-to-face. The main problem is that these employees need more attention, which requires heightened effort from the distributed team's managers.

At least at the beginning, providing attention and inspiration is more complex than with a co-located team. But, with practice, the distributed team's leader will hit their stride and understand how best to communicate with everybody.

In general, a distributed team's manager should push distributed team members to talk and give feedback about their status. The management should be aware of quiet individuals and take a proactive approach, aiming to avoid any delays in communication.

Also, keeping this in mind, when forming a distributed team, you should consider the members' soft skills and long-distance communication/operation abilities.

Another factor that could demotivate distributed team members is the feeling that they aren't properly informed about projects' processes. The manager is challenged to communicate with the team and provide updates on all relevant matters. Working in a distributed team needs a certain announcement style and for the manager to check that everybody's on the same page. The project manager should constantly check in with remote employees, verifying their understanding of the project's status, as well as their duties and tasks.

Miscommunication is one of the easiest ways to ruin a distributed team's workflow. Having a 1-on-1 check-in with team members weekly (plus a weekly all-hands meeting) will prevent such a situation.

4

Project and Process Management

In a distributed environment, the need for online management is crucial: real-time, high visibility project and process tracking enables all teams and developers to be fully engaged in the development process.

Addressing Challenges

Distributed Agile development teams can meet these challenges head-on by implementing Scrum, Kanban, Lean and Extreme Programming (XP) practices.

SCRUM

SCRUM maintains a customer-centric attitude and pledges that distributed teams will deliver the highest-priority project features first. Daily hand-offs or stand-ups between remote teams form a communication bridge and establishes a framework for frequent collaboration and synchronization. Furthermore, SCRUM Sprint reviews ensure that all delivered features are reviewed and that everybody receives feedback.

KANBAN

KANBAN is a very popular framework that's used to implement DevOps and Agile software development. The framework mandates full transparency of work, as well as real-time communication of capacity. A Kanban board is used to visually represent work items, which allows team members to see the status of each piece of work at any point in time.

LEAN

LEAN is a methodology that aims to optimize your organization's energy, effort, people, and resources – all in a way that provides value to the customer. Lean's key principles include:

1. Identify Value (specify value from the end customer's standpoint)
2. Map the Value Stream (when possible, eliminate steps that don't create value)
3. Create Flow (ensure that value-creating steps are tightly sequenced)
4. Establish Pull (let customers pull value from the activity that's next in the stream)
5. Seek Perfection (cycle through the process again until a state of perfection has been reached)

Extreme Programming (XP)

XP enables distributed software development teams to avoid quality and integration problems. For instance, because of continuous integration, remote teams get frequent integration points and can resolve problems as soon as issues arrive. Test-driven development achieves the simplest design that passes tests, while XP's refactoring practice strengthens software's evolutionary design and assists teams in resolving architectural challenges. Lastly, XP pairing reduces time wastage and provides continuous peer review.

Now, let's take a look at some specific work patterns that can increase the effectiveness of distributed dev teams.

- **Boot Camp**

  Boot camp brings all members of a distributed team together for the kick-off of a project. By attending the boot camp, all team members share an understanding of customer context and align on tooling, code standards, the definition of done, and initial architecture design. During this event, the crucial trust-building process begins.

- **Rotating Guru**

  Even after all team members kick off a project together, over time, their understanding, context, and trust will diminish. If a "rotating guru" regularly visits each location, they are able to infuse each distributed team with the home team's context and bring back insights to their own base – thus facilitating collaboration and "cross-pollination."

  During the guru's visit, they work at the location as a regular team member. They'll pair with local members whenever possible to observe their development style and figure out how they work. Not only does this foster understanding and communication, but it also creates a personal trust bond between the local team and the guru. For maximum effectiveness, the visit should be 3+ weeks long. A different team member should be assigned as the rotating guru per rotation; this will maximize cross-pollination benefits and reduce traveling burdens.

- **High Communication Modes**

  Face-to-face communication is one of the most effective methods of exchanging information. Co-located teams get to communicate in this manner daily, but it's more challenging within a distributed environment. Significant effort should be made to achieve a high level of engagement. Overlapping working hours should be established when remote teams are in different time zones. But, if this isn't possible, it will be necessary to create a dedicated program that offers special participation incentives. For instance, colleagues who opt-in to work earlier or later working hours could receive extra support.

  To maintain continuous, high-bandwidth communication between distributed teams, an assortment of instant messaging, video conferring, and desktop sharing tools should be used.

  Whenever possible, all locations should be involved in key events like Sprint planning and daily stand-ups. Teams might try to form their own local meetings, but it's imperative to have cohesive interaction across all distributed teams.

- **Remote Pairing**

  Co-located teams can benefit from the knowledge-sharing and collaboration that pairing brings. And it's even more important with distributed teams. Close collaboration is the most efficient way to build trust between members from different distributed teams.

  If there is a critical issue that needs to be resolved immediately, but it can't be handled entirely by one location's team, it needs to span over into the next time zone. In such cases, it's highly effective when one team member pairs up with a member of another distributed team for knowledge transfer. The understanding of the receiving team is much deeper, and trust is built. Pairings should be periodically rotated to ensure cross-pollination across team members.

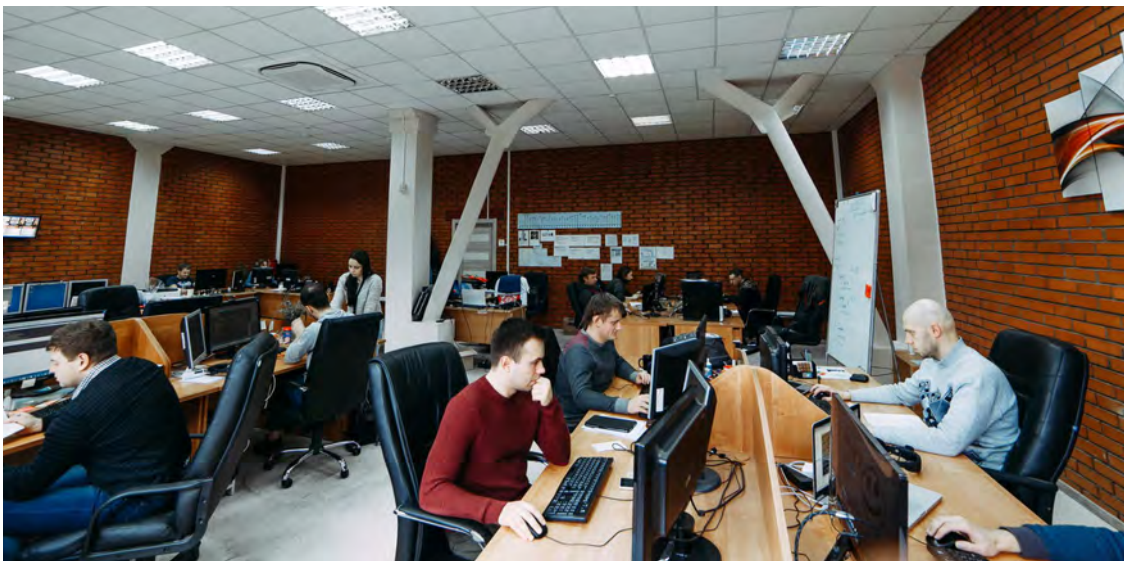
- **Develop a Shared Community**

  A shared community is a virtual community accessible by all distributed teams. It should contain a knowledge base that's used as a single source of information across all teams. Without a shared community, detailed information is stored in multiple sources, which then leads to confusion and wasted time. Some examples of things to include in a virtual community are:

Blog and Wiki posts

Online project management tool

Shared mailing list and common folder



- **Technology Alignment**

  In a trusting and transparent environment, technology, tooling, and development best practices should be aligned. However, this does not occur naturally in a distributed environment. You must make a deliberate effort to align critical elements and maintain alignment over time.

  Technology alignments

  The teams must establish frameworks used, coding standards, and the general promotion path of introducing new technology.

  Tool alignments

  The teams must align and standardize the integrated development environment, as well as commonly used tools like those for database queries.

  Engineering best practices

  A set of best practices should be established across all distributed teams – for instance, regarding TDD or refactoring.

Distributed teams face many communication boundaries that co-located teams don't have to worry about. But, by implementing the above strategies, you can keep time-zone, culture, and language obstacles to a minimum.

The Model Overview

Agile Software Development. This methodology calls for simple code, frequent testing, and the delivery of function, bite-sized application pieces once they're ready. Its main focus is to build a succession of parts as opposed to delivering one huge application upon completion.

Asynchronous Communication. This is communication that doesn't happen in real-time. It could either be due to the co-workers' time zone differences or fluctuations in scheduling.

Co-located Team. The majority of members in a co-located team are based out of one central office location.

Digital Workspace. This is a platform or integrated solutions set that lets teams access, complete, and manage their work from any location.

Distributed Software Development. It is a development process and it's associated environment in which individual components and modules are designed and implemented at different geographical sites.

Hybrid Team. This kind of team contains employees with diverse office attendance patterns. One team member might be fully remote, while another comes into the office on certain days. Some team members might be fully based in different geographical regions, while others are located at a central office.

Offshoring. This is the practice of re-allocating in-office duties to overseas workers. For instance, an organization might offshore its production warehouse or customer care center to another country. This can improve processes and save costs when done properly.

Outsourcing. When an organization hires a third-party vendor to provide goods and services or handle specific tasks, this is outsourcing. Such a solution might be necessary if the organization doesn't have available expertise or there is a lack of internal resources for a particular initiative.

Remote In-Sourcing. A remote team of experts that seamlessly integrates with the Client's in-house staff, giving Clients higher delivered value and flexibility.

Synchronous Communication. It is a real-time communication style that uses immediate back-and-forth dialog.

Main Providers

The main providers of distributed software development technology include:

Communication Tools

- Microsoft Teams
- Google Hangouts
- Slack
- Skype
- Zoom
- Google Meet
- WhatsApp

Collaboration Tools

- JIRA
- Asana
- Azure DevOps
- Notion
- GitLab
- ClickUp

Source Control Tools

- BitBucket
- SourceForge
- GitHub
- GitLab

A Closer Look: **GitHub**

GitHub is Git's most popular web-based repository hosting service, and it's considered an essential tool by most distributed software development teams. While there are available alternatives to Git (Subversion, Mercurial, TFS) and to GitHub (for instance, GitLab), this is still the most popular choice. GitHub offers a plethora of integrations with developer tools, a user-friendly interface, a huge public repository database, and features like issue tracking and pull request management. While GitHub isn't the only choice in its niche, it's tried-and-true.

A Closer Look: **Slack**

Just as how GitHub has become the leader of version control, Slack leads chat and has become the default communication solution for distributed teams. This seemingly simple solution enables teams to create public and private channels. They can also easily exchange files on this platform. While Slack's primary functionality is basic, there are plenty of bots and integrations that enable it to become a comprehensive distributed development tool. For instance, you can pipe in bug tracking alerts, integrate incidence response apps like Pingdom or Pager Duty, and even launch build-and-deploy events within Slack.

Main Points in Time in the Remote Work Journey

Remote work isn't new – it's just been growing in popularity in recent years. The COVID-19 pandemic triggered a huge migration towards remote work, but the trend in this direction triggered long before then. We've highlighted a few main points in time in the remote work journey.

1973

The term "telecommuting" was created by NASA engineer Jack Nilles in the book, The Telecommunications-Transportation Tradeoff.

1975

The first personal computers are made available.

1979

Five IBM employees were allowed to work from home as part of an experiment. By 1983, 2,000 of the company's employees were working from home.

1983

The internet is born.

Mid-1980s

J.C. Penny call-center employees are allowed to work from home.

1987

1.5 million Americans were telecommuting at this point.

1991

Wi-Fi is born.

1995

The first hackerspace, C-base, was founded in Berlin. Hackerspaces are considered to be one of the earliest models of coworking spaces.

2003

Offshore Dedicated Team™ was invented and registered.

2005

The San Francisco Coworking Space is created – it was the first official coworking space.

2011

Intetics trademarks the Remote In-Sourcing Model®.

2013

Slack is created.

2014

There are 26 fully remote companies in the US.

2016

Slack hits 4 million daily active users.

2018

There are 170 fully remote companies in the US.

2019

Zoom reports that 50,800 of their customers have over 10 fully remote employees, which was a 500% increase from 2017.

2020

The COVID-19 pandemic rapidly shifts the most common working model to remote. Hundreds of millions of people worldwide began to work remotely.

WHAT'S NEXT

?

The Market

In September of 2020, Gartner predicted that 75% of organizations that have distributed decision-making teams would meet and exceed their financial goals by the end of 2022. Diverse teams distributed across various geographies, ethnicities, and age groups are shown to have 12% higher employee performance than in homogenous teams.

As a response to COVID-19, numerous organizations were forced to distribute their teams. And the majority of industries adapted well to the absence of in-person collaboration. Upwork conducted a survey among 1,000 hiring managers; they found that a substantial percentage of company departments plan to continue having remote teams over the next 5 years. Remote work is now viewed as a strategic advance rather than a temporary measure.

2021 Worldwide Rates Comparison

WORKING WITH NEARSHORE OR OFFSHORE SOFTWARE DEVELOPMENT FIRMS CAN TYPICALLY SAVE A COMPANY FROM 40% TO 70% OF OVERALL COSTS.

Wherever the location – onshore, nearshore or offshore – development rates rose last year. Macro drivers were high demand for developers and greater use of outsourcing after the pandemic forced remote work and efficiency to be prioritized on the corporate agenda.

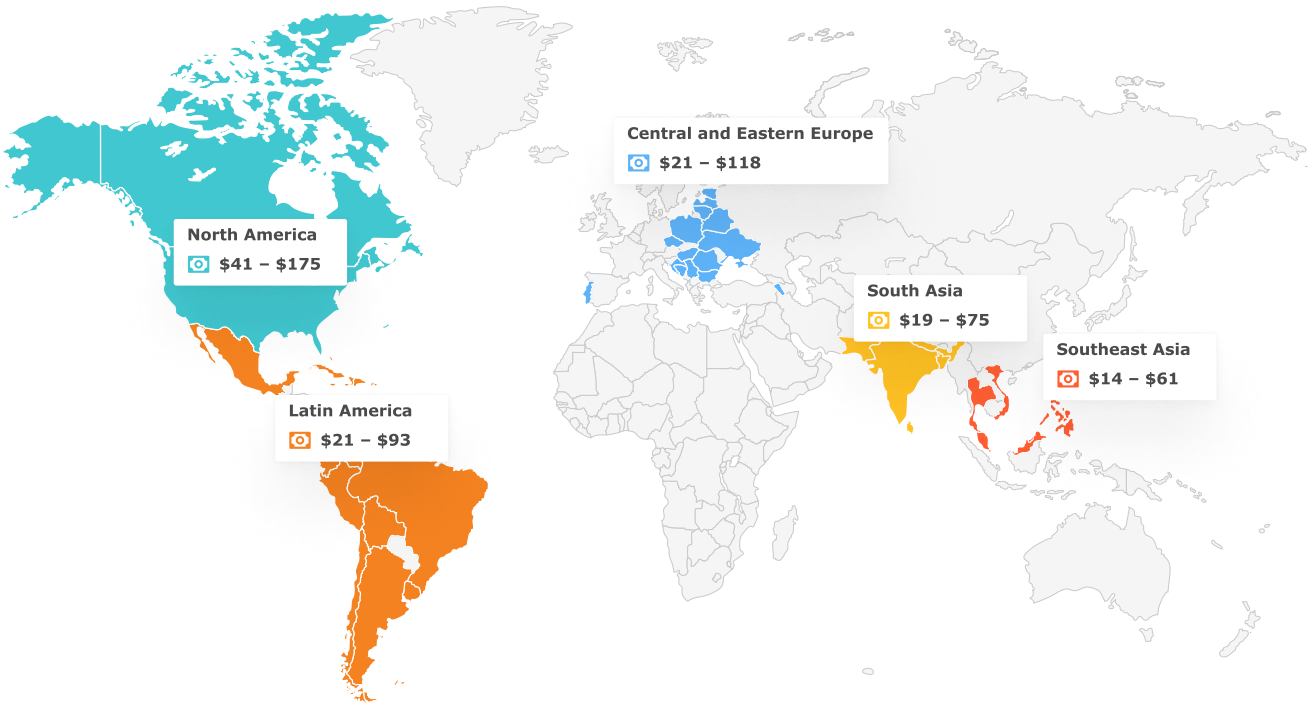
Rates trends: Based on our global network insights, rates increased by between 8% and 12% in 2020, varying by region. Expect further increases in 2021, although some firms may decrease margins to avoid rate hikes.

Record year: After the pandemic hit, more firms turned to outsourcing for a number of reasons, such as to control costs, to add more distributed resources, and to keep up with the competition. There was unprecedented interest in outsourcing as a result.

Regional dynamics: Patterns by geography held steady: Asian regions were still the least expensive, while Central and Eastern Europe continued to be a hotbed of growth. Nearshoring remained a desirable solution with its work-day timing and best-fit advantages.

Even with rising rates, software development outsourcing remains more costeffective than recruiting domestic resources. It's becoming a competitive imperative as more tech leaders adopt outsourcing and those who already have it in their staffing mix turn up the volume.

2021 GLOBAL OUTSOURCING RATES



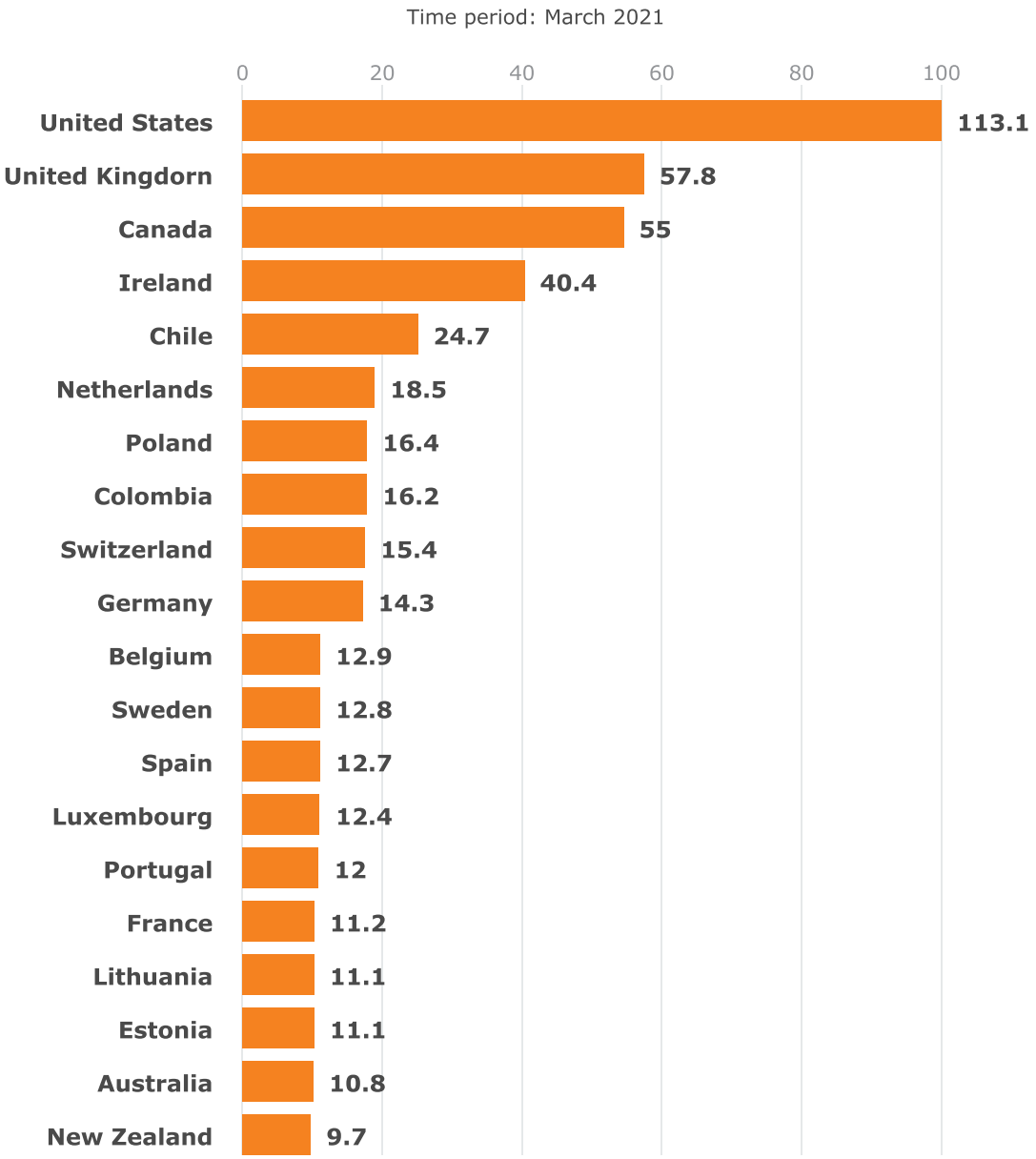
REGIONS ARE DEFINED AS:

- **Latin America**
  Argentina, Bolivia, Brazil, Colombia, Costa Rica, El Salvador, Mexico, Peru, Puerto Rico, Uruguay
- **Central Europe**
  Hungary, Poland
- **Far Eastern Europe**
  Armenia, Belarus, Ukraine
- **The Balkans**
  Bosnia, Bulgaria, Croatia, Serbia, Slovenia
- **South Asia**
  Bangladesh, India, Pakistan, Sri Lanka
- **Southeast Asia**
  Philippines, Vietnam

Source: Insider

Another important consideration about the distributed development industry is the cost of hiring developers from different locations. A distributed development team based in the US will be much more expensive than one in Ukraine. Above, you can see the average software developer salary in global hotspots. As you can see, there are different average salaries for on-premise and remote workers for each region.

20 COUNTRIES WITH THE MOST REMOTE JOBS ADVERTISED PER 100,000 OF THE WORKING-AGE POPULATION

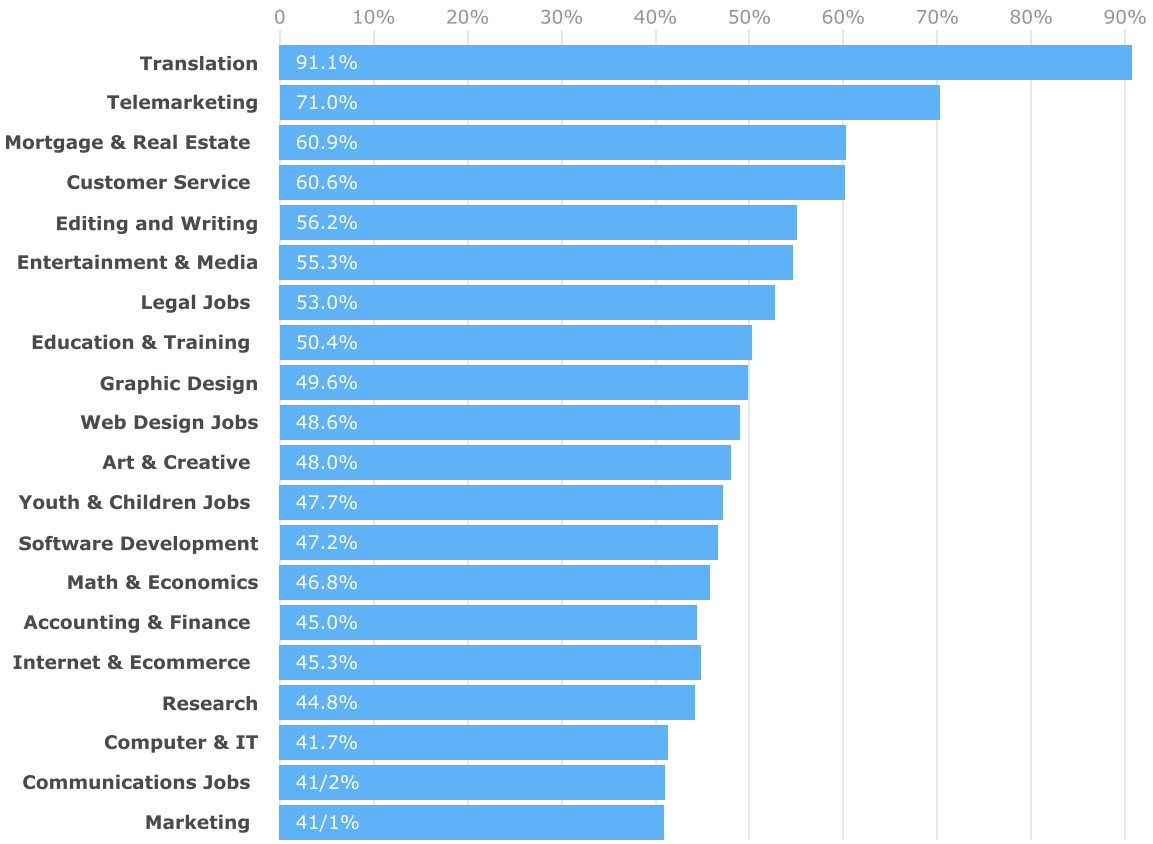


Data suggests that those based in the US have a greater opportunity to find non-office jobs.

Source: Insider

THE 20 INDUSTRIES ADVERTISING FOR THE HIGHEST PERCENTAGE OF REMOTE JOBS

Time period: March 2021

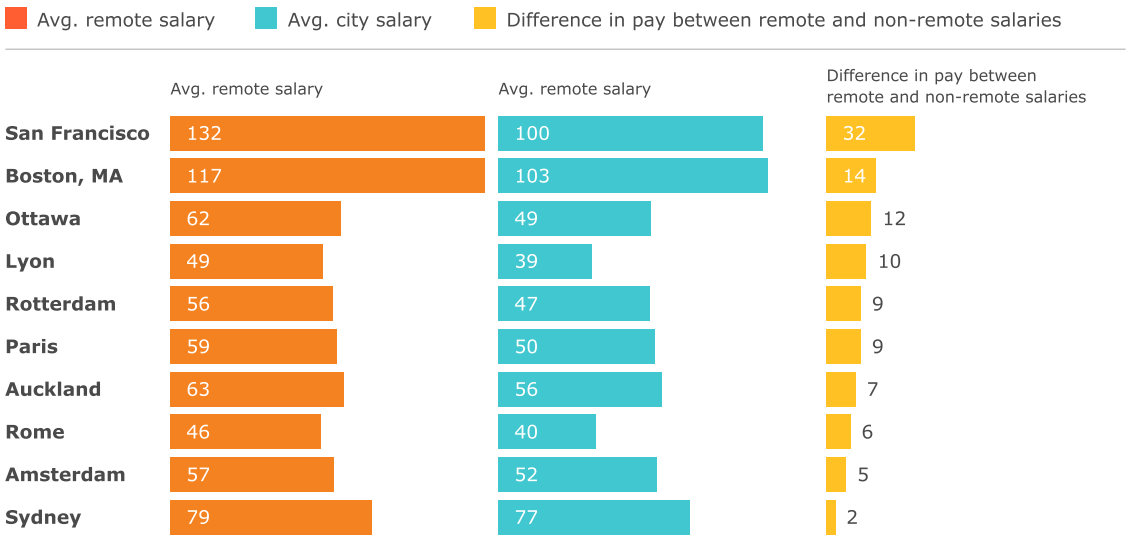


San Francisco is offering higher average salaries for remote work jobs being advertised.

Source: Insider

WHICH CITIES ARE PAYING MORE FOR REMOTE WORK?

Time period: March 2021



Source: Insider

How Does Setting up a Distributed Software Development Team Work

How does setting up a distributed software development teamwork? Forbes Tech Council has published insights on the process – we'll summarize their 8 key lessons:

1

Treat each team as a peer

Don't use words like "outsourced" or "offshore" – all this will do is sow division and make some teams feel like outsiders. Rather than having outsourced teams, you have a network of peer teams residing in different geographical locations.

2

Trust and Measure

The foundation of your organization's culture is trust – and if your teams are distributed, you'll have to make an extra effort to instill a sense of trust. You need to believe your developer's code, estimates, and overall commitment – but, what's more, you must also measure the elements that you trust. Some key areas that should regularly be measured include burnout charts, development estimates vs. reality, the team's velocity over time, and bugs found by QA. Such metrics apply to all distributed teams, no matter where they are located.

3

Hire for your company

Don't have a special interview process for distributed teams; each candidate, regardless of the location of the team, should go through the same interview procedure, and it should be RIGOROUS. Remember, you aren't hiring a contractor. You're interviewing and hiring a peer within your team network. Each node within your P2P network is of equal importance.

4

Try to make teams self-sufficient

Scrum teams should be self-sufficient, meaning that each one should have a few developers, a product manager, and at least one QA. Depending on the workload, product design could be a shared resource. Many companies prefer to keep Project Managers at HQ, but consider implementing on in a remote peer team. This way, they can be available to write acceptance criteria, do acceptance testing, and answer any questions QA and developers might have.

5 **Establish program management**

The program management office will work as a sort of referee within a P2P network of distributed teams. When you first make the switch to a distributed model, you might just have one Program Manager that reports to the COO directly and is independent of engineering. This manager makes sure the release is going smoothly, checks on dates and objectives, and quickly raises red flags when necessary.

6 **Establish a visual architecture team ASAP**

You'll need an architecture team that meets at least once a week to talk about design challenges. It's an effective way to drive architectural alignment across all distributed teams.

7 **Emphasize quality**

For all distributed teams, you should establish processes for unit, functional, regression, and security testing, as well as for code reviews. Furthermore, strive to fully automate testing within a CI/CD pipeline.

8 **Celebrate wins**

As we've emphasized before, building trust is important; members of distributed teams need to feel like they are part of the same company. Regularly schedule all-hands meetings, and share business news, welcome new employees, and celebrate wins.

Communication Paths in a Distributed Team

Who is who in a distributed software development team? And why does it matter? It's definitely necessary to define clear communication paths within a project team. Let's say you view a distributed team as a network in which each team member is connected with each other. In such a case, discussions would be lengthy, resulting in time wastage. On the other hand, if your communication is too tightly controlled and restrictive, it allows for no flexibility – thus impacting the team's trust and productivity (remember, we explained why trust is crucial in a distributed team).

So, the key to a distributed team role network lies in the middle of those two extremes. You must define communication paths – not with the goal of depriving

your developers of informal communication. Instead, you are creating a healthy balance, allowing the team to get the most out of both formal and informal communication styles.

A distributed team's communication path is controllable as long as everyone on the team understands it. Communication paths should be defined early on – and, as the project evolves, the paths can also evolve.

How do you begin defining communication paths? You'll start by mapping out member roles and responsibilities.

AGILE SCRUM ROLES

There are three roles here, which describe key responsibilities rather than job titles.

Product Owner

They are responsible for ensuring that the Agile team is delivering the most value. The Product Owner should understand the client, balance the needs of organizational stakeholders, and communicate to the development team what value needs to be delivered.

Scrum Master

This person holds everything together via a supportive leadership style. They help the Product Owner plan work with the development team, deliver effective learning, manage the backlog, and communicate value. And, on the flip-side, the Scrum Master helps the development team focus on outcomes, manage blockers, self-organize, and get to a "done increment."

Development Team

This doesn't just refer to engineers. The development team can contain writers, designers, programmers, and more.

STANDARD ROLES

Each distributed team will have certain roles that are the same from project to project. Such roles might include:

Client or sponsor

The person who is paying for the project. They communicate primarily with the product advocate or program manager.

Product advocate

Ensures that the sponsor's requirements are fulfilled. They communicate with the project manager, program manager, and sponsor, as well as with supporting roles.

Program manager

The main project coordinator; handles status reports, planning, risk analyses, action items, and regular meetings. They are the primary contact for the whole team.

Project manager

Coordinates schedules, manages deliverables, and produces status reports. In a smaller project, the program manager may also fill this role. Project managers are the primary contact for a specific site.

Development, QA, Technical Writing and UX Managers

Each Skill Set manager manages the personnel within their field of competence and helps with issue resolution as needed. These managers can make staffing decisions and notify the appropriate Project Manager – who, in turn, notifies the Program Manager.

System Architect

They are responsible for the software's design and architecture, as well as for how the software fits within its larger infrastructure. They communicate with the Program Manager, Product Advocate, Sponsor, Development Lead, and UX Lead.

STANDARD ROLES

Skill Set Leads

Each one manages the daily direction and workload of their skillset. They may also function as managers. They communicate with their team, as well as their skill set manager and Project Manager.

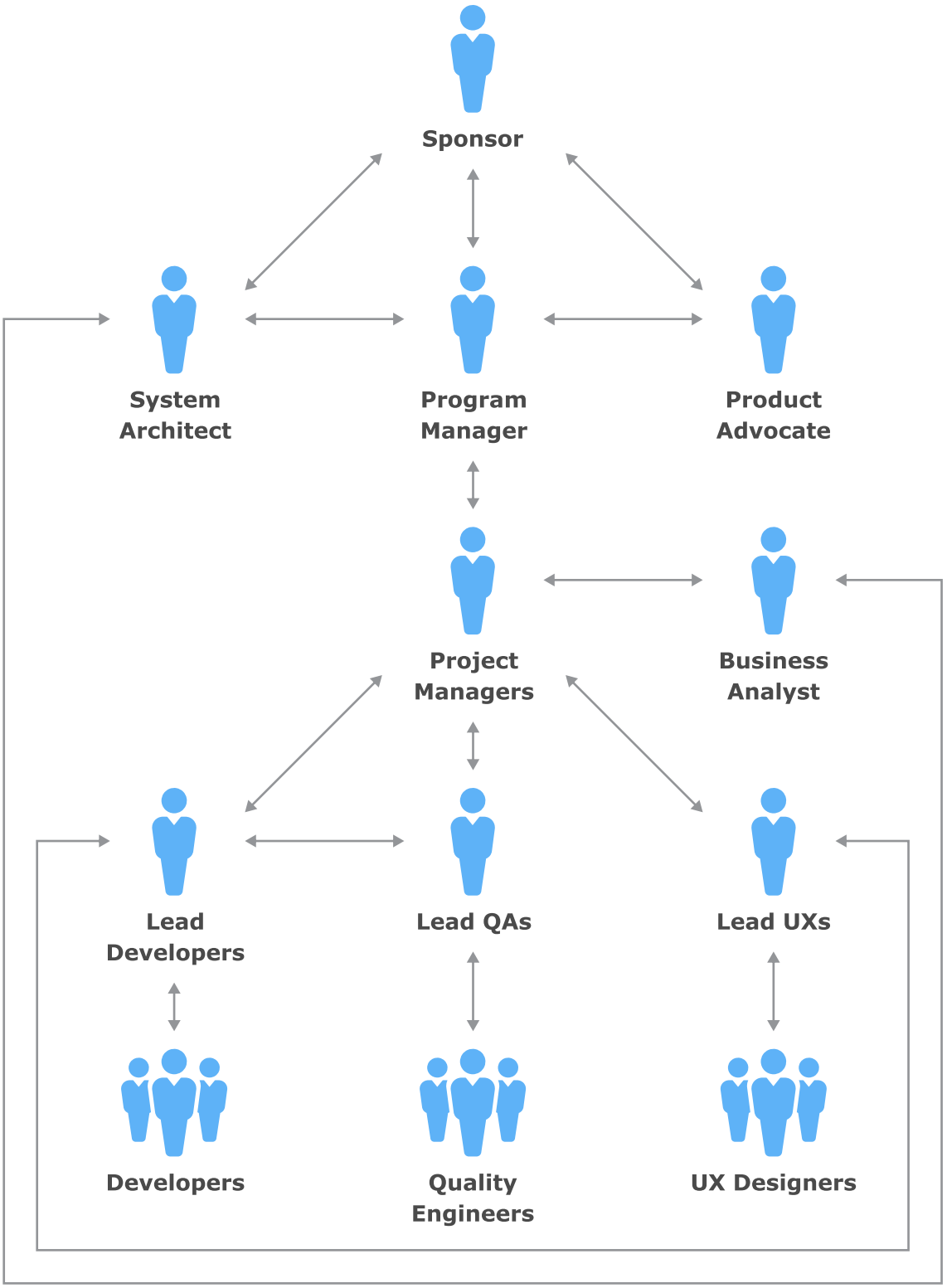
Line-level roles

These team members design and create the product. Developers write code and resolve bugs. UX designers draw wireframes, define visual elements, and write specifications. Technical writers create the product's written documentation, and Quality Engineers test code and flag bugs. They primarily communicate within their own team.

SUPPORTING ROLES

Some projects may need input from Subject Matter Experts, Business Analysts, and Technical Experts, as well as any representatives who have an interest in the final product. These individuals are usually called in when the project requires their approval or expertise. Supporting roles primarily communicate with each other, as well as with Skillset Leads and the Product Advocate. The Program Manager will be copied in on communications.

LARGE TEAM STANDARD COMMUNICATION PATHS



Diving into Technical Details

In order for a distributed team to be successful, its tech stack must be carefully selected. Included software should facilitate communication and collaboration across long distances. We'll walk you through the core components of a distributed team's tech stack.

Using the Cloud

One of the main drivers of the distributed development trend is the rapid transition to cloud-based platforms. Organizations can benefit greatly from using cloud-based services, largely due to scalability, cost savings, and shortened time-to-deployment.

The various Cloud platforms you'll encounter can be divided into 3 groups:

- **Traditional providers**

  With these, you can quickly get as much computing power or storage as you need. Developers will have many responsibilities before even touching a line of code: you must set up the server, a database and a runtime platform, and more. Some examples of traditional providers include Amazon Web Services, Google Compute Engine, Azure, and Rackspace.

- **Hosted PaaS**

  Value-added services are built on top of hosting. You get a lot more "out-of-the-box" solutions – for instance, fail-over, redundancy, and backups are built into the system. You also don't have to worry about server setup or configuration. Some examples include Heroku, OpenShift, Digital Ocean, and Linode.

- **Internal Cloud and PaaS**

  These tools create virtual internal clouds; even though they run in your internal data center, they still provide the on-demand, scalable resources that traditional providers give. Some examples include OpenStack, VMWare vCloud Suite, Mesos, and OpenShift Origin.

Continuous Integration (CI) Server

Continuous integration is a software development practice in which team members frequently deploy their work – most members commit a piece of code at least once per day. Some best practices include:

- Maintaining a single source repository
- Automating the build
- Making the build self-testing
- Each commit should build the mainline on an integration machine
- Testing is done in a clone of the production environment
- The build is fast
- Deployment is automatic
- Any team member can easily get a copy of the latest executable

CI servers execute a specific action on a repository upon receiving a commit hook. Some of the best CI servers for distributed teams include Hudson, Jenkins, Buildbot, Travis, Hubot, Weker, Distelli, and Capistrano.

Codebase in Git

The heart of the distributed development process is its version control system (VCS). At its most basic level, a VCS lets developers track each change made to a set of files and enables rollbacks. Distributed version control systems (DVCS) make it simple for multiple team members to work on one codebase simultaneously in different branches. These can be merged into a master branch. The basic process of a DVCS looks like this:

1. There is an agreed-upon project repository; it's typically on an internal server like GitLab or a public service like GitHub.
2. Developers all clone the main branch of the repository to their own local machines.
3. Developers create new branches for a specific feature.
4. The developer makes commits against their local copy.

5. After the feature is complete, the developer makes a Pull Request (also known as Merge Request), which holds all the code changes that are about to be made to the main branch. Other developers can review it and make suggestions.

6. Once code changes are peer-reviewed, they are merged to the main repository branch.

7. The other developers pull from the main branch and push in their own features.

8. The merged copy preserves the entire version history of each distributed copy.

Version control tools provide an easy way to peer-review code and make many simultaneous changes daily. Some of the best VCS for distributed teams are Git, Mercurial, GitHub, BitBucket, GitLab, and Gitorious.

Application Runs Locally in Development

One of the distributed development team's key tenets is that developers must be able to easily install and run the whole app on their local machine. This encourages flexibility and creativity, making development more productive and enjoyable.

One key tool here is Vagrant, which takes the recipes a developer created with their environment tool and provisions a virtual machine that matches the production environment as closely as possible.

VirtualBox is another great tool, enabling developers to run a full image of another OS on your own machine. And ngrok lets developers share apps running on their machine with other Internet users.

To replicate third-party APIs locally, we recommend using Canned or WireMock.

The Environment Should Be Automated in the Code

The environment in which your code is run should be modeled as code – not some separate black box. The key idea is that you need a recipe that lets your distributed developers recreate the environment at any time. Some key parts of managing the code's environment include:

- General configuration (setting up basic requirements, such as ensuring the directory exists, creating required users, specifying where log files are stored, creating security groups, etc.)
- Installing the application stack
- Installing and configuring backing services

Some of the environment automation and configuration tools you might encounter include OpsWorks, Fabric, Puppet, and CFEngine.

Infrastructure Must Be Monitored

The monitoring of infrastructure seems to be the most foreign element within the distributed development tech stack. In the past, if a disk filled up, the systems group would be alerted by the monitoring system, and they would fix it – there would be no need for the developer to be involved. Now, however, traditional admin functions are spreading to other parts of organizations – and, as such, there is a rising need for developers to view infrastructure monitoring tools. Besides providing critical alerts on failures, monitoring tools can pinpoint performance bottlenecks.

Some popular monitoring tools include New Relic, Scout, PagerDuty, Hubot, and loader.io.

Containerization

When an application is containerized, it and all of its dependencies can be packaged and shipped in a standard way that is identical for every platform. With containerization, you can package a container you built on your local development machine directly to a production server. There are many tools in this rapidly growing space, including CoreOS, Deis, Docker, Kubernetes, and Mesos.

Real-Time Chat

Real-time chat systems play a huge role in a distributed team's workflow. This is where team culture is formed and maintained and where most routine communication takes place. Some popular chat services include Slack, Campfire, HipChat, and Microsoft Teams. Chatbots can also be integrated within the real-time chat application, sending notifications and helping employees.

Use Cases and Main Applications

CASE 1

One of World's Top Customer Relationship Management Systems Scaled User Base up to 120+ Countries

Client

A software development company in California with an award-winning CRM system delivered via web app, mobile app, plugins and used in 120 countries.

Challenge

- The project's needs demanded infrastructure, resources management and top-level engineers with expertise both in custom software development and scalability.

Benefits and Results

- ★ The Client received dozens of the world's famous awards – CRM Market Leader, CRT Excellence Award, CRM Watchlist, and many others.
- ★ The CRM architecture was scaled to be used in 120+ countries and optimized with high-end technologies to meet the Client's business requirements.
- ★ Long-term partnership with Intetics and 10+ years of project development.

Quick Facts

- 10+ years of continuous cooperation
- The Offshore Development Center managing all aspects of the project was created
- CRM scalability – users from 120+ countries
- Top Remote In-Sourcing Team® members received an opportunity to advance their career in the Client's organization

Technologies

Open Source stack / PHP / AngularJS / JavaScript / React / RESTful / SOAP Web Services / Elastic Search (FTS) / Ruby / DB2 / Oracle / MSSQL / MySQL / Mongo DB / RabbitMQ / Python / Go / AWS

Case Study

Download Full Version

CASE 2

Remote In-Sourcing Team® Significantly Enhanced Cloud Solution by Moving to Microservices and Elasticity

Client

The Client's international website for recruiting with a huge infrastructure operates in 19 markets across the world to connect finance professionals with hiring companies.

Challenge

- The Client experienced maintenance, support, and performance issues of the global web portal with millions of daily users and required team reinforcement.

Benefits and Results

- ★ The Client migrated with zero downtime or negative effect on operations.
- ★ The risk of malfunctioning, collapsing due to human error, or hosting issues was eliminated.
- ★ 1.7M user accounts along with CVs were carefully migrated without data leakage.
- ★ The platform became empowered by AWS technologies with enhanced features and better services.

Quick Facts

- 99.99% website availability
- Monolith was split into 31 microservices
- Zero deployment downtime is a result of following CI/CD for all services

Technologies

AWS / Java 11 / Spring / Quarkus / Angular / React / Python / GoLang / NodeJs / Solr / Elasticsearch / Redis / Terraform / Docker / Nginx / WordPress / SQL Server

Case Study

Download Full Version

CASE 3

Intetics Remote In-Sourcing Team® Created and Supports One of the Biggest Health Portals in the USA, Serving 5m People a Year

Client

A Healthcare company that unites a group of clinics. Was founded by a successful physician who recognized that existing practice management and marketing tools did not work together to help practices, practice groups, and hospitals develop their businesses

Challenge

- Finding the right experienced Software vendor in HealthTech.

Benefits and Results

- ★ Creation of the digital business from scratch: from idea to execution.
- ★ The Client`s previous attempts to build the system failed due to the previous vendors' skill shortages.
- ★ The new system serves 5,000,000 people on an annual basis, with this number continuing to grow.
- ★ The first version of the system was delivered in under 10 months after project kick-off.

Quick Facts

- 10 years of continuous cooperation
- Designed and implemented 100% of the Client's products
- The Client can concentrate on their medical business rather than the technical details

Technologies

AWS / Microservices / .NET Core / React / SignalR / WebRTC / Entity Framework / Swift / Kotlin / HTML5 / Java / JavaScript / TypeScript / PHP / WordPress / Kazoo / Google Speech to text / Web sockets / OpenCV

Case Study

Download Full Version

Examples of Other Companies That Have Distributed Their Teams

Continuous integration is a software development practice in which team members frequently deploy their work – most members commit a piece of code at least once per day. Some best practices include:

zapier

Workflow tool **Zapier** has over 1,000 app integrations and allows the transfer of data between these apps – and it lets you automate tasks from your devices. 80 people work at Zapier from across 13 countries. Zapier's co-founder, Wade Foster, stated that the distributed development model is a better way to work, as it allows you to hire smart people from anywhere in the world.

doist

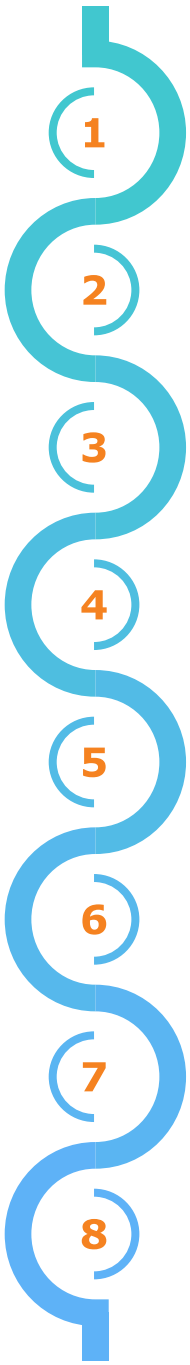
Doist is the distributed company behind Todoist, a top-ranked productivity app. Over 50 people work at Doist, and they're located across 20 different countries. The COO of Doist, Allan K. Christensen, believes that the distributed development model is the future of work and should be adopted by other companies.

AUTOMATTIC

Automattic, the company that developed Tumblr, is entirely remote – it has over 1,150 members in 72 countries. The firm's founder, Matt Mullenweg, stated that the distributed model prioritizes productivity over working hours and leads to a more ethically responsible business.

Standards Applied

While there are no official distributed development industry standards, there are plenty of best practices to ensure your team is as effective as possible. These include:



1. Applying agile practices like short iterations and small releases to increase task meaningfulness.
2. Providing a moderate to high degree of autonomy.
3. Making daily stand-ups and Sprint planning meetings to emphasize regular feedback.
4. Establishing roles and expectations upfront but letting core norms uniform naturally within the team.
5. Keeping teams as small as possible.
6. Selecting sites with some overlapping work hours.
7. Using multiple forms of information and communication technology while placing emphasis on groupware, desktop sharing, instant messaging, and teleconferencing solutions.
8. Acknowledging that an ideal virtual team should span cultural, functional, and organizational boundaries.

Industry Resources

- **IEEE Software Magazine**

  This resource delivers cutting-edge information on rapid technology changes, including the transition to geographically distributed development. Some other topics include the education of software professionals, standards, architectures, and programming environments.

- **ACM Digital Library**

  This is a compilation of all ACM publications, which aim to promote the highest software development standards and recognize technical excellence. There are plenty of publications related to distributed software development.

- **Remote Work Association**

  Here, global business leaders of virtual teams network, learn, and teach – with the goal of fueling the future of location-flexible work.

Authorities to Follow

Leaders in the remote team software space:

- Slack and Microsoft Teams (team collaboration)
- Zoom and Microsoft Teams (video conferencing)
- Trello and Jira (project management)
- Google Drive, OneDrive and Dropbox (file management)
- GitHub, GitLab (version control)

Leading companies with a distributed workforce:

- Automattic
- Zapier
- Buffer
- GitLab
- Time Doctor
- DuckDuckGo
- Doist
- Stack Overflow

Certifications

Leading Remote Teams Certificate – eCornell

This certificate program improves the taker's ability to effectively manage remote teams. It consists of five modules, and each course is 2 weeks long.

Using Git for Distributed Development – Coursera

This course focuses on using Git for distributed development of open-source software. Git helps geographically varied developers coordinate work in a practical way.

Global Software Development – edX

This free course gives developers the organizational and technical skills necessary to practice software engineering within a globally distributed team.

Distributed Team Healthcheck

Distributed software development teams are definitely here to stay, and you may already be thinking of introducing this team model. But managing a globally distributed team is challenging – even more so if you don't use the right remote work tools or strategies. The systems you implement need to give your distributed teams access to everything they need for their jobs, including communication and remote access. We've prepared a "health check" to help you assess distributed work systems and determine which one is the best fit for your company.

User Productivity

- Does the solution work under bad network conditions with low bandwidth and high latency? How about offline?
- Can your users get personal and corporate access with one set of peripherals and a single device?
- Does your solution have a responsive UI experience?

Security

- Will your users have access to corporate resources through a trusted and safe operating system?
- Does your distributed work solution protect against app vulnerabilities, OS vulnerabilities, insider threats, etc.?
- Does your solution prohibit malware from accessing corporate network resources?
- Is the solution compliant with your client's requirements, as well as financial or government regulations?

Manageability

- Have you taken the cost and operational overhead into account, including cloud/network/data center costs?
- Can the IT infrastructure (firewall throughput, VPN, etc.) support this new load of connected devices?
- As your company grows and opens up new team locations, can the solution ensure a consistent user experience?

Further Reading

- **Practical Remote Pair Programming**

  Learn tips, techniques, and best practices for productive collaboration within a distributed development team.

- **Geographically Distributed Agile Teams**

  Learn about how geographically distributed teams can adhere to the agile philosophy, how distribution affects tooling choices, how it relates to scaling factors, and more.

- **Distributed Teams: The Art and Practice of Working Together While Physically Apart**

  This book will help you create tighter teamwork and better communication across your entire team – regardless of where they're located.

- **Strategies to Improve Collaboration for Distributed Agile Teams**

  This video gives techniques to address the collaboration and engagement challenges that distributed teams may face.

- **Agile Development – Working with Agile in a Distributed Team Environment**

  Learn how to tailor Agile to fit your team, optimize the size and mix of your distributed team, set up pair programming, understand cultural differences, and more.

- **Why Distributed Software Development Teams Work Infinitely Better**

  Learn about the main benefits of Distributed Team model over the traditional one from the experience of an industry expert.

- **What is Distributed Agile Software Development**

  The most unbiased and informative definition of Distributed Software Development.

- **How to build a successful distributed software development team**

  Take a look at the process of hiring only the best professionals from all over the world for your software development team.

Top Tips for Working With Distributed Development Teams

It's crucial to maintain a structured work process and effective communication when your team members are located across the globe. This list of best practices will help you build a transparent and healthy working relationship with your developers.

Establishing Robust Communication

- Schedule regular meets (backlog refinement, daily standup, Sprint planning, Spring review, retrospective, etc.). This keeps everybody on the same page regarding risks, hurdles, and work progress.
- Make sure each meeting has a clear agenda, and stick to it as closely as possible.
- Consider time differences when scheduling meetings. This is why ensuring hours overlap is important.

Task Management

- Create a Product Backlog within your task tracking tools and make sure all team members have access. Sort the items by priority, with the highest-priority tasks at the top.
- Be consistent in how you describe tasks. Good requirements should be feasible, correct, prioritized, necessary, verifiable, and unambiguous.
- Run collaborative estimation sessions in which everybody involved in development participates. Otherwise, you might find that estimation accuracy will fluctuate significantly.

Regular Feedback

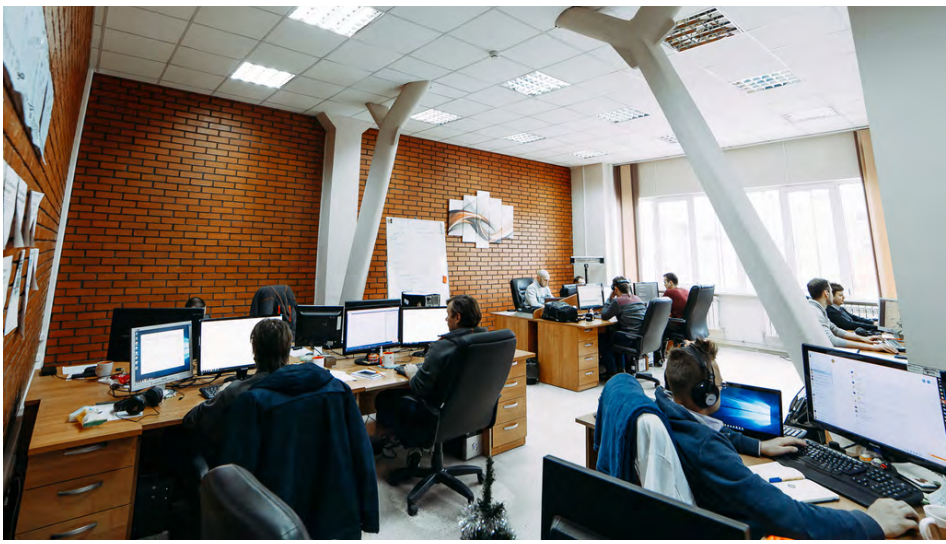
- Give regular, honest feedback on team and individual performance.
- Give regular feedback on deliverables; address what could be improved, but don't forget to specify what is good in the code and functionality.
- Hold retrospective meetings regularly.
- Set up multiple performance measurement metrics.

Building a Healthy Culture

- Hold video chats regularly.
- Promote team values such as Courage, Focus, Respect, Commitment, and Openness.
- Use any opportunity for knowledge transfer.

Summary and Conclusions

Distributed software development can be a boon to your organization – from cost-cutting to talent pool effectiveness and time savings. However, distributed teams are more likely to experience transparency and communication challenges than a co-hosted team. But by using Agile development processes and collaborative software, you can foster trust between your teams, no matter where they're located.



intetics
Where software concepts come alive™

INTETICS MEANS YOUR SUCCESS

Toll Free: +1 (877) SOFTDEV
US: +1 (239) 217-4907
DE: +49 (211) 3878-9350
UK: +44 (20) 3514-1416
Email: intetics@intetics

www.intetics.com