intellias

Testing Automotive Software

Trends, Challenges, and Best Practices

Table of contents

Executive summary

In-vehicle software has been in constant motion for years, transforming, disrupting, and steering the automotive industry. In going digital, the automotive industry sees few limits. As it steps into the world of connectivity, manufacturers are tuning cars with better dashboard displays, improved control and safety systems, advanced infotainment systems, and other features that make driving more comfortable and prove that the future of autonomous vehicles is possible — and near.

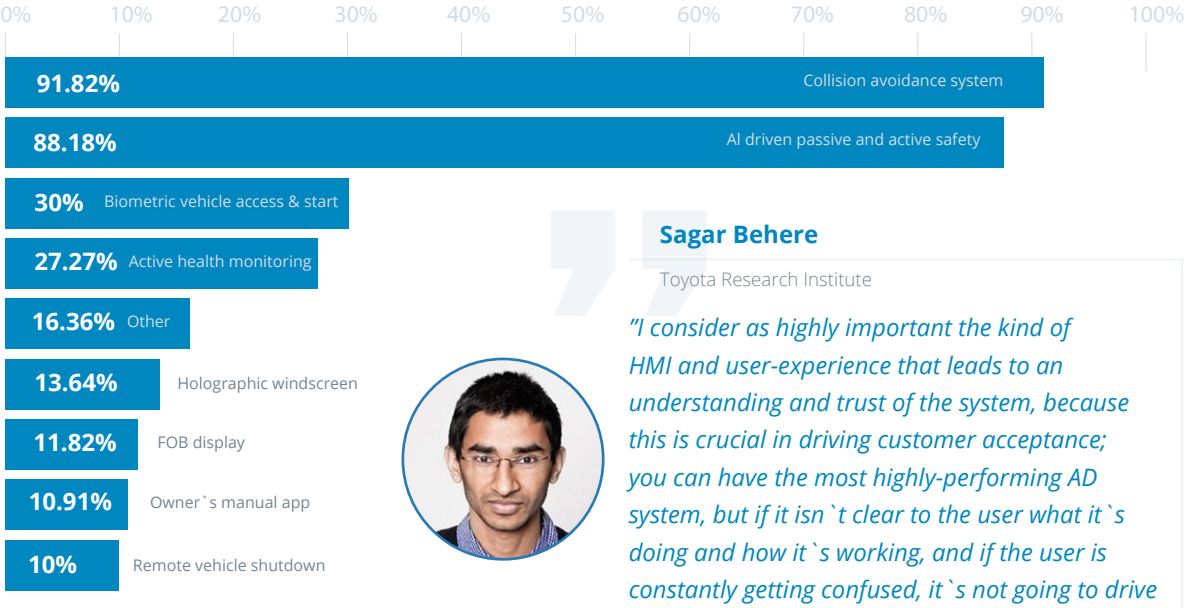
Automotive software is the only way to upgrade cars to a new level of efficiency without raising costs. More precisely, in-vehicle software allows for more complex systems and plays an indispensable role in increasing a car's functionality. Major changes to the cars can be traced to improved circuit analysis, behavioral modeling, driver assistance systems, and many other areas of optimization.

In a single year, auto manufacturers can produce a combined 70 million cars globally. Meanwhile, a single connected vehicle requires 100 million lines of code according to General Electric. Delivering a glitch-free car requires special attention to all phases of the project — testing is critical.

Automotive software development trends

Automotive software development trends have always been contingent on customer expectations. The more innovation end users expect, the more businesses need to tweak their models and adapt them to the demand. Car electronics are being shaped by the features customers want to have at their fingertips. Self-driving technology is the first of these features, and it's setting the direction for breaking new ground.

As self-driving technology evolves, consumers are beginning to express interest in vehicles with higher degrees of automation and more advanced machine learning systems. When asked what trend will significantly affect the automotive industry in the coming years, **Sagar Behere** from the **Toyota Research** Institute confidently names collision avoidance systems and warning systems, explaining that they are the most expected by customers. Such systems are also receiving the greatest investment at present.



Graph 1.

Sagar Behere

Toyota Research Institute

"I consider as highly important the kind of HMI and user-experience that leads to an understanding and trust of the system, because this is crucial in driving customer acceptance; you can have the most highly-performing AD system, but if it isn`t clear to the user what it`s doing and how it`s working, and if the user is constantly getting confused, it`s not going to drive acceptance of that system."

Other trends include AI-driven passive and active safety together with biometric vehicle access. Behere says that autonomous vehicles with these technologies have the potential to enter the market in ten years. But first, it's essential to test AI-driven systems in real driving conditions.
Together with security and safety, testing and validating complex sensor combinations remains the top concern among OEMs according to the 2018 Automotive IQ Report.

Functional safety is a system challenge that needs to be addressed early on in the design process of the system.

Automotive IQ Report, 2018.

Software development processes: V-model vs Agile

It's essential to distinguish between two methodologies for managing a software development team: the V-model and Agile. While at first glance it might seem the V-model is a better choice because we're talking about the automotive industry, Agile is more effective, as it's more flexible and allows for change as part of the testing process.

So what's the difference between these two methodologies, especially when testing is concerned? Let's investigate further.

V-model

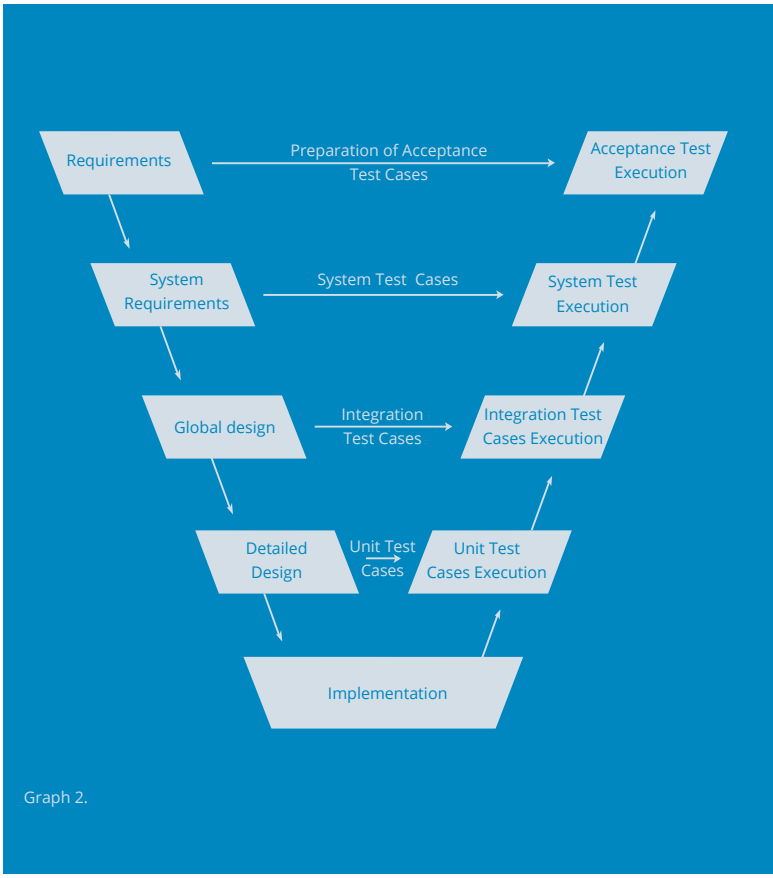
The V-model describes a development process that proceeds in the form of a V. Even though it's possible to adjust the requirements when following this methodology, doing so necessitates going through the whole V-model again.

V-model

Using the V-model, you're expected to have a clear-cut plan to perform software testing for each component. Testing should happen alongside software development to reduce the chance of discrepancies. Put simply, software testing professionals are involved from the beginning, not only after the software development process is completed.

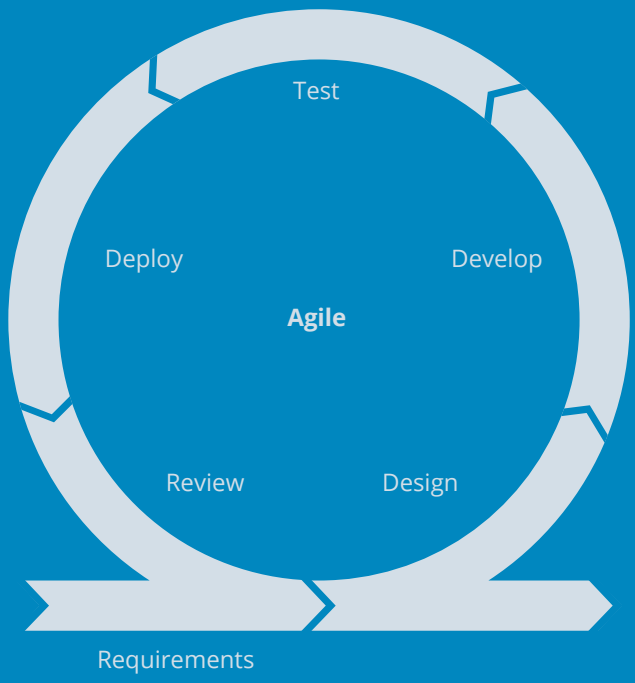
There are many variants of the V-model; the most common involves four levels of testing with different objectives:

- **Component (unit) testing** searches for defects and verifies the functionality of software components (modules, programs, objects, classes, etc.) that can be tested separately.
- **Integration testing** tests interfaces between components, interactions between different parts of a system such as the operating system, file system, and hardware, and interfaces between systems.
- **System testing** checks the behavior of the whole system as defined in the project scope. The focus of system testing is verifying specified requirements.
- **Acceptance testing** validates software with respect to users' needs, the project requirements, and business processes to determine whether the system meets the specifications.



Graph 2.

The obvious advantage of the V-model is that verification and validation are carried out side by side. This model aims to develop a quality product whose parts are tested and corrected immediately at each phase of the project. However, there are also some drawbacks to this model. By its nature, the V-model is rigid, so if anything needs to be tweaked partway through, all the requirements and test documents need to be updated. While it can be used for short-term projects, this methodology is less effective when applied to developing software that needs constant maintenance and repair.



Graph 2.

Agile

Agile software development, in contrast to the V-model, is a methodology in which a program is developed gradually in iterations, improving step by step from a minimum viable product. Agile development does not demand long-term planning and is thus more flexible and open to changes. Agile processes require regular cooperation between team members and discussions of intermediate results.

Agile model

The most popular Agile framework, Scrum, calls for working in two-week sprints, during which the team is supposed to deliver something meaningful that will be improved in the following sprints. At the end of the sprint, the team demonstrates new functionality to the client.

The greatest advantage of Agile is that the team has a product backlog with all the requirements, but work is organized in a way that the client can change anything from the backlog whenever necessary. In Scrum, a daily meeting is held to go over what's been done, who's doing what, and if there's anything preventing the team from performing well.

Agile testing, in contrast to V-testing, is a continuous process, not a sequential one. Even though there's less structure and planning to Agile testing, the final product responds to the end user's needs and pains.

When choosing between these two methodologies, project managers need to consider the context. When the requirements regularly change, Agile is preferable.

As a rule, cars are built step by step and the production process is waterfall-like. However, each part, especially software, can be broken into smaller pieces and be developed in phases. This is where Agile methodologies and Scrum in particular come in handy.

With Scrum, the whole project is broken into parts to make a work breakdown structure. As in the waterfall process, parts are developed consecutively, and each has a due date. However, the team responsible for developing a given part can apply any methodology they like within the time allotted. The manager of the whole project is the one responsible for controlling the development of all parts and ensuring the project is finished on time, on budget, and on value.

Challenges of Automotive Software Testing

Regardless of the chosen project management methodology, automotive software testing entails numerous challenges.

Challenge 1:
Because the testing environment involves physical objects, it needs to be prepared in advance. Project managers might need equipment from a car — or even a car itself. Examples of test environments include cars with embedded infotainment systems, vehicle identification, data loggers, and video cameras.

Challenge 2:
Testing automotive software requires branded hardware. Sometimes, issues may arise on account of necessary equipment being incompatible. Firmware validation usually calls for additional effort, as firmware testing is carried out for each unique brand configuration (with its own attributes and quirks). There's also a substantial risk of bricking hardware units if incorrectly configured software is installed.

Challenge 3:
Additionally, troubles may appear when the car's vehicle identification number (VIN) comes into play. The VIN is the code that identifies a specific car, acting like the car's fingerprint. Because two working vehicles cannot have the same VIN, registrations should be updated and maintained to enable the use of different web services such as for traffic and parking information.

Challenge 4:
Testers can find it difficult — and often impossible — to test software in an emulated environment. And even if the environment is successfully emulated, there may be restrictions. A vivid example is road signs that look different in a navigation setup in the lab and during a real test drive. Such discrepancies make testing challenging.

At the same time, field testing is an indispensable part of development, as many functions should be tested only during a test drive. These include:

Car positioning

A car's position should be correctly recognized in conditions difficult to emulate: tunnels, urban canyons, multi-level underground parking, ferries, etc.

Traffic

Some systems avoid heavy traffic to provide the driver the optimal route. Navigation should handle different traffic information providers, such as online and radio, depending on their availability, and switch between them without issues.

System stability

Field testing gives the opportunity to check system stability when all components are integrated (navigation with connected services, climate control systems, radio, etc.).

Persistence of settings

Persistence of settings after the car is shut down and restarted.

Generally, the cost of software failure in the automotive industry is high, and testing is expensive and complex. This motivates software development companies to establish reliable processes from the beginning and avoid risk. The automotive industry is subject to standards including IATF 16949, AEC-Q100, and AEC-Q200, which have to be followed during testing.

Best practices in automotive software testing

Automotive software testing has been around for years, and many best practices have proved their utility along the way. Let's consider types of testing, test processes, and bug reporting.

Types of testing

There are many types of testing, and your choice will depend on which criterion you're classifying. For example, to find out if the requirements are followed, positive and negative testing can be applied:

Positive testing

Do things the way the user is supposed to. Example: Create a route and follow it.

Negative testing

Test edge cases to make sure the program doesn't break. Example: Deviate from the set route.

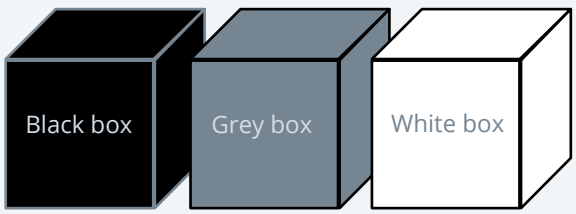
Different types of testing focus on different levels:

- **Unit testing –** Developers create tests for the smallest possible parts of code, called units or functions.
- **Integration testing –** Developers or testers verify whether everything works correctly when these units are combined.
- **System testing –** Testers verify that the system works overall. Example: Find a location through search functionality, start a route to it, and check traffic and guidance along the way.
- **Acceptance testing –** Users verify that the product works correctly and validate the process. Example: Customers perform test drives.

The choice among manual testing, automated testing, and semi-automated testing is also significant.

Automated testing is used when performance testing is mostly automated for the navigation system, while semi-automated testing can be applied when firmware validation includes both manual and automated stages.

Based on your knowledge of the inner structure of the program and access to the source code, there are three types of testing you can perform:



Graph 3.

- **Black box –** Test the software as a user with no knowledge of the program's inner structure.
- **Grey box –** Test the product with some knowledge of its inner structure. Example: Test the performance with mock server data and modified config values, or use test data prepared in advance, such as recorded routes.
- **White box –** Test the software with access to all source code.

Whenever you need to test what the program does, functional and non-functional testing comes into play.

- **Functional testing** checks what the program does — for instance, whether it successfully receives online traffic data and behaves correctly according to the requested route.

- **Non-functional testing** checks how the program does this.

Non-functional testing, in turn, may comprise:

- **UI testing.**
  Does the program look like what we expect or what the designers described? Example: Is there an animated 3D junction view before complicated intersections and exits?

- **Usability testing.**
  Is the program convenient? Example: Check voice guidance that accompanies on-screen instructions.

- **Security testing.**
  Is the program secure? Can hackers break in and steal or tamper with data? Do we transfer and store our sensitive data safely? Example: Try logging into personal car accounts on test devices and mobile apps with user credentials or a security token.

- **Configuration testing.**
  Does our system work well in all supported environments (on all devices, operating systems, etc.)? Example:
  Test navigation builds on a variety of firmware.
- **Performance testing.**
  Is our program fast enough? Example: Load the system in complicated traffic conditions.

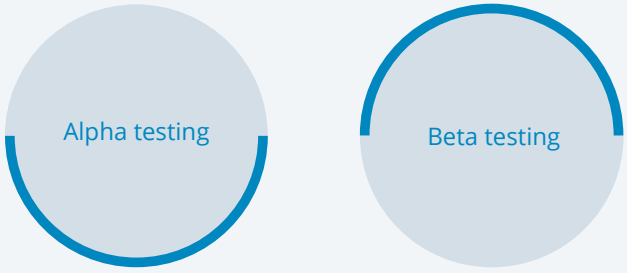
- **Recovery testing.**
  Example 1: When the application is receiving data online (satellite view, traffic info, online search), unplug the connecting cable (D-Link) or deactivate the SIM card. Example 2: Start a customer map update via SD card or USB the same way as an E2E User. Unplug USB and plug it in again. Check that the map update continues from the point of interruption.

- **Localization testing.**
  Do speakers of supported languages understand the program interface?

Based on who conducts the testing and where, we can distinguish between alpha testing and beta testing.

- **Alpha testing** is when the company allows the testing team to use the company's equipment — like with E2E field testing, for example.

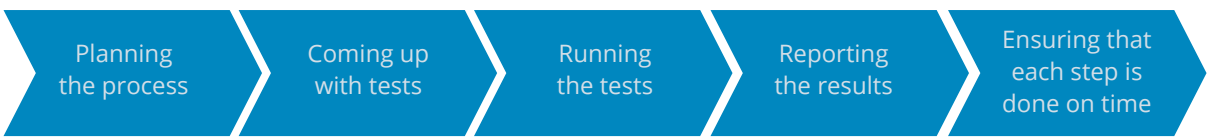
- **Beta testing** is when the client or a limited group of users run tests on their own equipment and under their own conditions, such as conducting test drives on the customer's side.



Graph 4.

Testing process

The testing process usually involves several stages:



Planning tests

To manage the testing properly, it's important to ask yourself a few questions to avoid trouble in the future.

- **What do we need to test?**
  Planning a test starts with collecting general information about the system under test, its parts, priorities, non-functional requirements, etc.

- **Who does the testing?**
  Define the team, its members, their skills, their capacities, their roles, etc.

- **What do we need to perform the testing?**
  What equipment, hardware, and software is required?

- **Who does what, when, and for how long?**
  Each team member has their own testing priority and should know when they need to do what. Team members should provide time estimates for their part of the testing process.

- **When will we consider the program ready for release?**
  Once all the tests are performed successfully and there are no more than 10 low-priority bugs left.

Designing tests

Test design is a complex process that should be carried out with precision. In automotive software testing, special attention should be paid to equivalence partitioning, boundary value analysis, and pairwise, cause-effect, and error guessing.

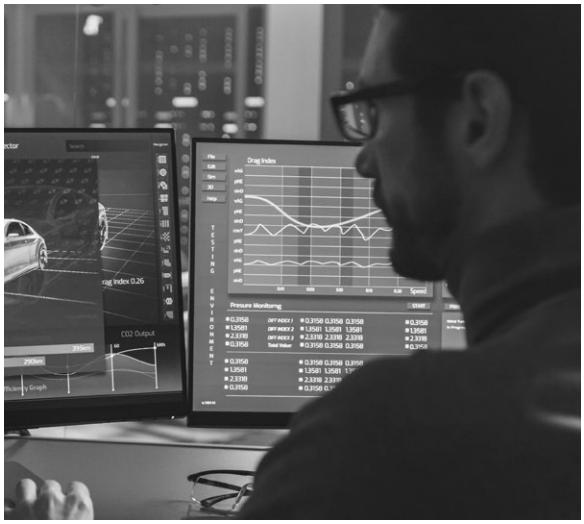
- **Equivalence partitioning.**
  Consider each of the program parameters and determine subsets of values so that the behavior of the program is similar for each subset. Test one value from each subset: if it works fine, the rest will also work fine; if it doesn't, the rest will also cause a problem.

- **Boundary value analysis.**
  Test the boundaries and values immediately before and after those boundaries and identify what is most likely to cause bugs.

- **Pairwise.**
  Combine the values from several parameters so that they could meet in at least one of the tests. This reduces the number of all possible combinations to test while still being certain there are no issues caused by the definite combinations of values.

- **Cause-effect.**
  If you have some constraints in the requirements or your test ideas, make sure to come up with at least one test for each.

- **Error guessing.**
  Test anything you come up with outside of the previously mentioned test design techniques.

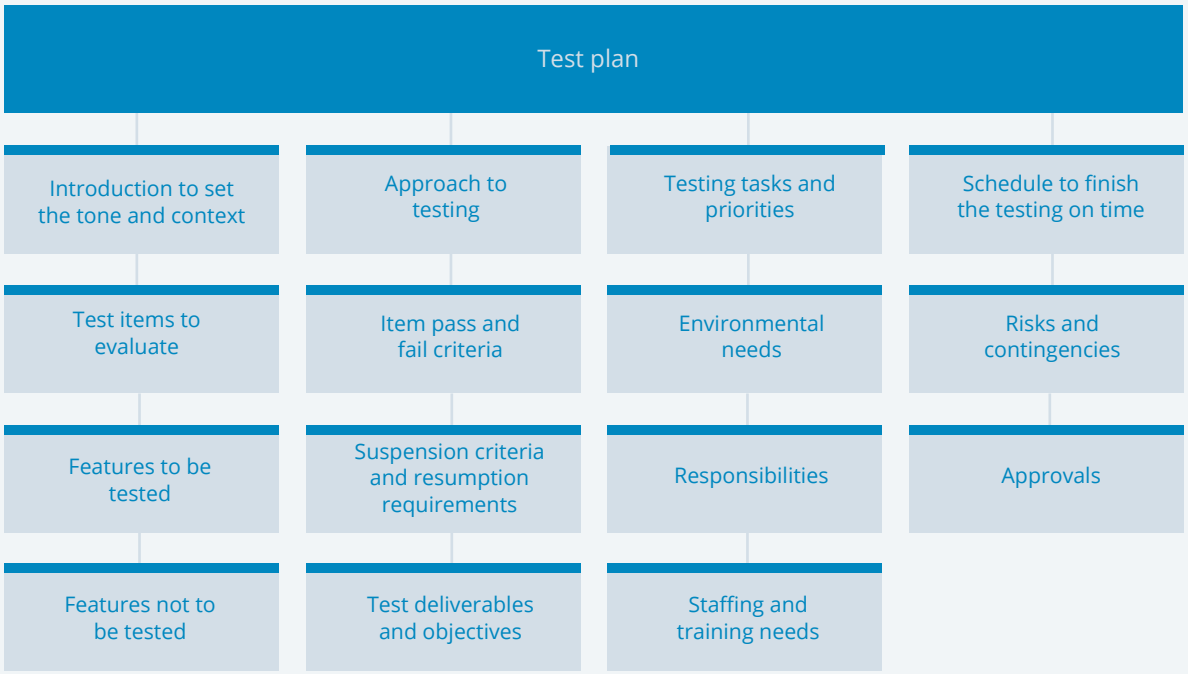


Test documents

It's essential to have the following test documents to perform testing.

Create a checklist or a list with short descriptions of each test to be performed. Your checklist should include the following:

- **Test cases** – Describe each item from the checklist more precisely as a set of preconditions (what needs to be done or obtained before the test), steps, and expected results.
- **Test suite** – A set of test cases grouped by logic.
- **Test plan** – A document describing the scope, approach, resources, and schedule of the intended test activities. According to IEEE 829, a test plan should consist of:



Graph 5.

In turn, the test summary report should contain:

- **Environment details**
- **Link to the executed test suite**
- **List of findings or reproductions**
- **Failed and blocked test cases with an explanation or linked ticket number**
- **Test verdict – An overall summary of the build status and the most critical findings that require attention (for example, no online connectivity over the external SIM card).**

Reporting bugs

Bug tracking systems, or project collaboration tools, store descriptions of program features, improvements, and bugs. They provide common access to information so every team member can reach it, allow for building reports to aggregate information and make estimates, and show how long developers spend on tasks.

Jira is one example of a project management system. Whenever testers find a bug, they can describe the error and put it into Jira. Bug report parameters should include a summary of the bug with a short description of the issue, the environment it was found in, and the preconditions for recreating it.

Test cases

Intellias has run many test cases in the automotive industry, including mobile app testing, testing of predictive routes, E2E regression of customer map updates, and testing of maps according to regional variations. In every case, you should pay special attention to many factors.

Mobile app testing

Testing automotive mobile apps requires you to define the scope of testing, automated tests, manual tests, and test data.

- **Scope of testing** is to verify that a driver can set a route in a mobile app and successfully send it to the car. Received routes can be applied in the car. Data on a mobile and embedded system is stored and synchronized.
- **Automated tests** cover security verification and synchronization of user data.
- **Manual testing** on the emulated environment is run to cover functional and non-functional aspects (failover and recovery testing). Field testing will cover basic use cases and stability testing.

E2E testing of predictive routes

To test predictive routes, it's good to stick to the following plan:

- **Scope of testing** — Verify that the driver's frequent routes, home, work, and favorite locations are stored in the system as expected. The driver should receive route suggestions at appropriate times (for example, suggestions for the route home in the afternoon). Predictive data should be preserved and updated in the system and on the server.
- Perform **manual testing** in the emulated environment during initial testing stages, followed by **field testing** under real conditions, which is the highest priority. **Functional testing** should be combined with **non-functional** in order to check the performance and correct work on different car configurations — for example, on older and newer versions of hardware and on brand or model-specific configurations.
- **Smoke** and **regression tests** should be run to verify the overall functionality and ensure changes to the code do not break the existing codebase. Develop and run a set of new feature-specific E2E test cases.

Compared to mobile app testing, testing of predictive routes would also require a test case design together with test execution and reporting.

Conclusion

Automotive software development never stops. Driven by high customer expectations, the industry is experiencing innovative changes, especially in the realm of software. As customers expect higher levels of machine learning and automation, special attention is being paid to improving car electronics.
However, the problem with testing remains. As automotive software testing should almost always be performed under real driving conditions and requires specialized hardware, project managers may experience many issues. Without a full-service offering, OEMs face substantial risks. To overcome the challenges of automotive software development, choose the right project management methodology and type of testing. Then define the optimal testing process, create a testing plan, design your tests, and choose the documents you need in order to follow testing best practices. Intellias has deep experience in automotive software testing and has got you covered. With a hand from our team, you'll be able to release a quality automotive software product.

About Intellias

Since 2002 Intellias has been a trusted software engineering partner for OEMs and Tier 1 vendors from Germany, the UK, Japan, and South Korea. Intellias' software development expertise and accumulated knowledge within the automotive industry has helped create adaptive and AI-based automotive software solutions for connected and autonomous driving.

Intellias has the competence, scalability, and experience to build safe automotive software used by global car manufacturers.

For more information, reach us at info@intellias.com or visit us in our offices in Berlin, Krakow, Lviv, Kyiv, Kharkiv, Odesa, and Ivano-Frankivsk.