ChatGPT

Automotive Software Update Process Workflow

The software update process for modern vehicles is a multi-step pipeline that spans from initial development to the actual on-vehicle installation and validation. In general, it begins with **development and integration** of new code (features, fixes, calibrations) by suppliers/OEMs, followed by **rigorous testing and verification**, then **packaging** of the update into deployable artifacts, **secure delivery** to vehicles (often over-the-air), and finally **installation and monitoring** on the vehicle. Standards like ISO 24089 and UN R156 explicitly call for end-to-end processes including planning, verification, risk management, and traceability throughout the update life-cycle [1] [2]. In practice, many organizations adopt automated pipelines (CI/CD) so that every code change is built, tested (unit, integration, hardware-in-the-loop, safety/cybersecurity checks), and staged for deployment [3] [4].

- **Planning & Scheduling:** Updates are planned well in advance, often tied to vehicle variants and feature schedules. OEMs and suppliers coordinate roadmaps and synchronize multi-ECU releases.
- **Development & Integration:** Engineers implement features or fixes and merge them into the codebase. A supplier's release process typically "integrates functional extensions, modifications and bug fixes into software components, verifies them, and then integrates the verified components into the complete software" [5]. Tools like version control and automated CI ensure changes are tracked and built.
- **Testing & Validation:** Automated test suites and simulations run at each integration. This includes safety compliance checks (ISO 26262, AUTOSAR) and security scans (static/dynamic analysis) to catch issues early [4] [1]. Continuous testing at scale (including hardware-in-the-loop and system testing) ensures the update meets quality and regulatory requirements before release [6] [7].
- **Release Packaging:** Once verified, the update is packaged. This may be a full firmware image or, more efficiently, a differential "delta" package. Differential updates are constructed by comparing new vs. old firmware; in one example, a "diff" file was typically <10% of the full size [8]. Sonatus reports that delta algorithms can reduce update size by up to 95% [9]. Each package is cryptographically signed and often encrypted to ensure authenticity and integrity.
- **Delivery & Campaign Management:** The packaged update is uploaded to the OEM's server or cloud platform. It is then scheduled and delivered to vehicles – commonly via cellular networks, Wi-Fi, or in-factory/dealer links. A central management system (Software Update Management System, SUMS) tracks which vehicles/models should receive the update, orchestrating large-scale rollouts in campaigns [10] [11]. Progress and success rates are monitored in real time. Standards like UNECE R156 mandate that all such updates be logged, approved, and traceable [2].
- **Installation & Monitoring:** In-vehicle, a secure "OTA manager" (often on the gateway ECU) coordinates the update. The telematics control unit (TCU) or gateway downloads and verifies the package [11]. Each target ECU runs an update agent that validates the image (checking signatures) and then installs it. A common strategy is a dual-bank (A/B) update: the ECU writes the new firmware to an alternate flash bank while keeping the old version running. After successful download, the ECU switches to the new bank (usually at next startup), ensuring there is always a working fallback [12] [13]. Once installation is complete, the vehicle reports success (or failure) back to the OEM. Throughout this execution, diagnostics are collected and any errors trigger built-in rollback or retry procedures.

Update Development and Release

In automotive practice, **development and release** of updates follows structured lifecycles. Updates often originate from multiple suppliers, so their integration must be carefully managed. As one guideline explains, “functional extensions, modifications and bug fixes are integrated into software components according to the release plan. The verified components are integrated into the complete software, and the integrated software is verified as planned” [5]. In other words, suppliers integrate changes in small cycles (often agile-style) and synchronize them at system-level milestones. The final *release artifact* may include not just binary code but also calibration data, documentation and (if needed) hardware units. The scope of each release is predefined and validated before delivery [14].

Modern automotive teams leverage **CI/CD pipelines** to automate much of this. Code changes are merged into a central branch, automatically built, and subjected to a suite of checks for style, safety, and compliance [3] [4]. Continuous integration ensures that each change is quickly tested, catching integration bugs early. When a software increment is ready, it is tagged for release and assembled into a deployment package (e.g. a container or firmware image). Automated processes enforce automotive standards (ASPICE processes, ISO 26262 compliance, coding guidelines) and generate audit trails [4]. For example, code scans for vulnerabilities and license compliance can run on every commit, while approval workflows gate when a build is qualified as a candidate release. This disciplined pipeline ensures that when engineers “create a release item” (the final deliverable), it meets all functional and quality requirements [5].

Once the release package is built, it is handed off to the software update management team. At this point, further validation (system tests, regression tests, vehicle-in-the-loop tests) may be run on the final package. Only after passing all verification steps is the update approved for deployment. In regulated markets, each update must also be documented for type approval: UN R156/ISO 24089 require evidence that each deployed update has been checked and authorized [1] [2]. Thus the *development & release* phase involves not just coding and compilation, but thorough multi-level integration and sign-off processes before the update leaves the control of the developers.

Key points: Automotive updates originate in multi-vendor development cycles and use rigorous integration pipelines. Features and fixes are merged, tested, and integrated into full ECU or vehicle software releases [5]. Continuous Integration/Continuous Delivery (CI/CD) tooling automates builds, safety/security scans, and prepares deployment packages [3]. Final releases include not only code but calibration data and documentation, and are delivered only after all functional and compliance tests are complete [14] [7].

Validation and Approval

Before any update reaches a vehicle, it must be **validated** and formally **approved**. Validation encompasses functional testing (unit, integration, end-to-end), safety verification, and security evaluations. Automotive standards enforce this: for example, ISO 24089 “outlines the key processes and functions…including verification and validation, and risk management regarding safety and cybersecurity” [1]. In practice, this means the new software is tested under all relevant scenarios (including Hardware-In-the-Loop tests, stress tests, and regression suites) and checked against safety requirements (ISO 26262) and industry process models (ISO/IEC 15504 – Automotive SPICE). A CI/CD pipeline typically automates much of this testing, with tools that perform static code analysis, security scans (SAST/DAST), and even automated compliance checks for ASPICE/ISO26262 [4]. Each code change “undergoes comprehensive testing: simulated vehicle system

testing, hardware-in-the-loop validation, performance testing and integration testing" [6], ensuring high confidence in the update's correctness and safety.

Approval refers to the formal authorization to release the update to vehicles. In the automotive context, this is a serious matter. Regulations now require OEMs to have a certified Software Update Management System (SUMS) that governs approvals. UNECE Regulation 156 explicitly "makes the approval of software updates mandatory for all vehicles" and ties it to type approval [2]. In effect, every update must be signed, recorded and deliverable only to compliant vehicles. Practically, this often means *digital signing* of the update package: when an update is approved, its code (or delta file hash) is signed with the OEM's private key so that the vehicle will accept only authenticated updates. The update path must also respect all safety and cybersecurity protocols; e.g., encryption in transit, anti-rollback checks, and secure boot mechanisms are commonly enforced. Industry best practices (endorsed by authorities like NHTSA and standards bodies) include encrypting the update payload, using TLS and certificates for communication, and logging all update events [15] [16].

In summary, **validation** ensures an update works correctly and safely, while **approval** ensures it is secure, authorized, and auditable. Both are mandated by process standards and regulations. Automakers typically maintain a formal workflow with multiple "quality gates": only after an update passes all verification tests and obtains managerial sign-off (or SUMS certification) is it pushed to vehicles. Any update that fails validation is either revised or blocked. This rigorous approach – combining exhaustive testing [7] [4] and regulatory compliance (UN R156/ISO 24089) – is necessary to prevent safety issues or cybersecurity breaches in the field.

- *Validation:* Functional checks (simulation, HIL, integration) plus safety and security verification. Automotive CI/CD pipelines embed automated ASPICE/ISO26262 checks and vulnerability scans into every stage [4]. Post-update, vehicles often self-test to confirm integrity (with fallback if checks fail).
- *Approval:* Regulatory and internal sign-offs. OEMs must use an approved software update management process (SUMS), as required by UNECE R156 [2]. Each update package is digitally signed and time-stamped, ensuring only authentic, untampered updates install. Approvals are tracked and logged for audit.

Packaging and Delivery

Once an update is verified and approved, it must be **packaged** and **delivered** to vehicles efficiently and securely. Packaging involves assembling all modified assets into a deliverable bundle. This bundle might be a single large firmware image or, more often, a set of smaller units (e.g. individual ECU images, or container images in modern architectures). Differential packaging is common: the OEM compares new and old firmware to create a "delta" or "diff" file containing only the changes [8]. Such delta packages can be an order of magnitude smaller than full binaries, dramatically reducing download time and data costs [8] [9]. In practice, tools automatically generate and optimize these deltas; for example, Sonatus reports that its updater can apply delta algorithms to cut OTA payloads by up to 95% [9]. All packages are cryptographically sealed – signed and optionally encrypted – to guarantee authenticity and confidentiality during transmission.

Delivery is typically over the air. The OEM's server or cloud holds the update, and the vehicle (via its telematics control unit or gateway) will fetch it over a mobile/cellular network or Wi-Fi when convenient [11]. Update distribution is organized into campaigns: the update management server determines which

vehicles or VIN ranges should receive the new software and when. Large fleets are updated progressively (e.g. batch by batch) to monitor for any unexpected issues. According to Bosch, at the manufacturer side "software artifacts need to be made available as a download, and software updates must be assigned to eligible devices. When dealing with a large number of devices, it takes carefully managed campaigns to orchestrate the associated rollouts. Progress of the software update process must also be monitored from the start" [10]. In other words, the system schedules downloads, pushes notifications or auto-downloads to vehicles, and tracks success/failure rates in real time.

Throughout delivery, security remains paramount. The update connection is secured (typically via TLS) and the payload is verified on download. Many systems use a central OTA backend and a vehicle-side agent that performs checksum and signature checks before accepting the package. If an update fails integrity checks or installation, it is rejected. The vehicle then may report the failure back to the OEM. Together, careful packaging and managed OTA campaigns ensure that the right vehicles get the right software, with minimal bandwidth and maximum reliability [11] [10].

Notable practices:
- **Delta vs Full packages:** OEMs often deliver "diff" updates (<10% of full size) whenever possible [8]; otherwise full images are used.
- **Delta optimization:** Advanced tools (e.g. Sonatus Updater) automatically generate optimal deltas to "minimize update sizes by up to 95%" [9].
- **Secure delivery:** Packages are digitally signed. The vehicle's telematics unit downloads from the cloud, verifies the signature, then alerts the owner (or auto-proceeds) to install [11].
- **Campaign management:** Cloud platforms coordinate rollouts, schedule deferred installs (so updates happen at convenient times), and monitor progress [10]. This end-to-end orchestration — from cloud to ECU — is essential for large-scale OTA.

Vehicle Integration and Update Execution

The final stage is **integrating and executing** the update on the vehicle's hardware. Once the telematics unit or gateway has received and authenticated the update package, the in-vehicle update manager takes over. Typically, an **OTA Manager** service (running on the gateway ECU or a central computer) orchestrates distributing the update to each relevant ECU [17]. NXP and others recommend placing this manager on the central gateway ECU because it "only runs OEM-trusted software" and sits at the secure bridge between the vehicle's internal network and the outside world [17] [18]. From there, the manager pushes the update to target ECUs over the vehicle's bus (CAN, Ethernet, etc.), ensuring compatibility (by checking ECU IDs and version tables [19]) and decrypting as needed for legacy ECUs lacking hardware security.

Each target ECU typically follows a **two-step installation process**. First, the ECU writes the new firmware to a secondary memory bank or storage while its original firmware continues running, as described by the "A/B" approach [12]. For example, many modern ECUs have dual flash banks: one holds the current code and the other is updated in the background. This allows the vehicle to operate normally during the download/install. Once the ECU has successfully written the new firmware, it is configured to boot from the updated bank on the next restart. Critically, since the original code still resides in the other bank, the ECU can instantly revert to it if anything goes wrong [12] [13]. This A/B scheme ensures there is "always a working firmware" and virtually eliminates the risk of bricking a module during update [13]. In systems without dual banks, updates may be done in-place, but this requires the ECU to be non-operational during flashing (and thus is less desirable for safety-critical nodes).

In practice, most vehicles update when they are parked or in service mode. The user typically gets a notification (“Update ready – install?”). Upon acceptance, the TCU signals the update manager to proceed, then the vehicle may prompt the driver to keep the car off or connected to power (to avoid battery drain). The update manager then directs each ECU’s bootloader to begin the flashing. Each ECU verifies the received image (checksum/signature) before committing it. If all ECUs successfully install the update, the vehicle prompts a reboot sequence (e.g., restarting each ECU or the entire vehicle). The new software then boots up. The vehicle may perform a self-check or calibration after reboot, and then report success to the OEM backend.

If any ECU fails to install correctly or validate its new software, it typically remains in its old state (thanks to the backup copy) and the update manager logs the failure. Many systems can then retry or roll back to the previous state. For example, an ECU with a dual-bank design can simply remain using the primary bank and skip the failed update. The central OTA manager may even have a backup of the old firmware (especially for ECUs without dual banks, as NXP suggests storing a backup copy in external flash) to restore a corrupted module [20].

Throughout the update, the system enforces safety: critical ECUs (like powertrain or braking controllers) are often updated separately from non-critical ones to avoid simultaneous outages. The entire process is also secured: only encrypted, signed images are accepted, and communications are authenticated. Overall, the vehicle-side update execution is a carefully choreographed sequence that updates each subsystem in turn, verifies success, and falls back gracefully on failure.

Key aspects of execution:
- **OTA Manager location:** Placed on the secure gateway ECU for best isolation [17]. It holds metadata (ECU IDs, serials, versions) and coordinates the update distribution [19] [18].
- **Download & Verify:** The gateway/TCU downloads the update (cloud→vehicle) and checks its signature and metadata match the vehicle’s configuration [11] [19].
- **Dual-bank Flashing:** ECUs use A/B partitions so the new code is written “in the background”. Only after all writes succeed does the ECU switch to the new bank on reboot [12] [13].
- **Reboot & Confirmation:** The vehicle reboots ECUs or itself into the new software. Each updated ECU self-verifies its software on startup. Success/failure is logged and reported back to the OEM.
- **Fail-safe:** If an update fails, the vehicle retains the old firmware. Many architectures allow automatic rollback (ECU stays on the old partition). Some systems even support recovery by loading stored backup images [20].

By following these steps across development, release, and vehicle installation, automakers ensure updates are delivered safely and securely throughout a vehicle’s life [1] [2]. Proper adherence to standards and automated workflows reduces risk and helps maintain vehicle safety, compliance, and customer satisfaction.

Sources: Authoritative industry sources were used throughout. For example, one industry guideline details supplier release steps (integration of fixes, verification, delivery of software and data) [5]. OEM platforms (e.g. Sonatus Updater) describe cloud-to-vehicle OTA orchestration with logging and delta packs [11] [9]. Regulatory summaries emphasize ISO 24089/UN R156 compliance for secure updates [1] [2]. These and other sources inform the above workflow and best practices.

---

1 ISO 24089: Vehicle Software Update Engineering Standard | TÜV SÜD
https://www.tuvsud.com/en-us/industries/mobility-and-automotive/automotive-and-oem/autonomous-driving/iso-24089-standard-for-automotive-software-update-engineering

2 UN Regulation 156 Assessments - Automotive Software Updates | TÜV SÜD
https://www.tuvsud.com/en-us/industries/mobility-and-automotive/automotive-and-oem/autonomous-driving/assessment-of-automotive-software-updates

3 4 6 CI/CD for automotive software development | CircleCI
https://circleci.com/blog/ci-cd-for-automotive-software-development/

5 14 zvei.org
https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/april/Best_Practice_Guideline__Software_Release/Best-Practice-Guideline-Software-Release-engl-2016.pdf

7 Managing Secure and Compliant Automotive Software Releases
https://www.cyient.com/blog/managing-secure-and-compliant-automotive-software-releases

8 12 13 17 18 19 20 nxp.com
https://www.nxp.com/docs/en/white-paper/Making-Full-Vehicle-OTA-Updates-Reality-WP.pdf

9 Automotive OTA Update Management | Sonatus Updater
https://www.sonatus.com/products/updater/

10 Software updates over the air - Developer Bosch IoT Suite
https://bosch-iot-suite.com/knowledge-center/use-case/software-updates-over-the-air/

11 15 16 What is OTA in automotive? Over the air updates explained. - Rambus
https://www.rambus.com/blogs/ota-updates-explained/