Testerský slovník pokračovanie

🔹 **Canary release**  
Postupné nasadzovanie novej verzie len na malú časť produkčných serverov/používateľov, aby sa overilo, že nová verzia funguje, skôr než sa rozšíri na celek.

🔹 **Blue-green deployment**  
Strategia nasadenia, kde existujú dve identické produkčné prostredia (blue a green). Nová verzia sa nasadí na nepoužívané prostredie a po úspešnom otestovaní sa “prepne” na používateľov.

🔹 **Chaos engineering**  
Prístup, pri ktorom sa do systému cielene zavádzajú chyby (výpadky, latencia, zlyhania služieb), aby sa overila jeho odolnosť a schopnosť zotaviť sa.

🔹 **Service virtualization**  
Simulovanie správania závislých služieb (napr. externé API, mikroservisy), ktoré ešte nie sú dostupné alebo sú nákladné, aby testy mohli bežať izolovane a rýchlo.

🔹 **Contract testing**  
Testovanie dohôd (kontraktov) medzi službami – overenie, či poskytovateľ aj konzument API dodržiavajú spoločné špecifikácie.

🔹 **Data-driven testing**  
Technika, kde sa rovnaký testovací scenár spúšťa s rôznymi vstupnými dátami (napr. z CSV/tabulky) bez potreby písať duplicitný kód.

🔹 **Keyword-driven testing**  
Štruktúra testov, kde sú kroky definované pomocou „kľúčových slov“ s mapovaním na implementované funkcie, čo uľahčuje údržbu a čitateľnosť skriptov.

🔹 **Model-based testing**  
Generovanie testovacích scenárov automaticky z formálneho modelu (napr. stavového diagramu) systému, čím sa zabezpečí lepšie pokrytie.

🔹 **Orthogonal array testing**  
Matematická technika na redukciu počtu kombinácií vstupov pri zachovaní dobrého pokrytia, často používaná pri testovaní veľkého množstva parametrov.

🔹 **Usability testing**  
Testovanie z pohľadu reálnych používateľov, zamerané na to, ako intuitívne a pohodlné je používateľské rozhranie, typicky s využitím scenárov a pozorovania.

🔹 **Accessibility testing**  
Overovanie, či je aplikácia prístupná aj používateľom so zdravotným znevýhodnením (napr. čítačky obrazovky, klávesové skratky, kontrast farieb).

🔹 **Security testing**  
Testy, ktoré odhaľujú zraniteľnosti (XSS, SQL injection, CSRF), kontrolujú autentifikáciu/autorizáciu, šifrovanie dát atď.

🔹 **Performance profiling**  
Podrobná analýza, kde dochádza k úzkym miestam (CPU, pamäť, I/O) v aplikácii, často s použitím profilovacích nástrojov.

🔹 **Pair testing**  
Dve osoby (tester–tester alebo tester–developer) spoločne pri jednom počítači identifikujú chyby a diskutujú o scenároch, čo zvyšuje kvalitu a vedomosti tímu.

🔹 **Shift-right testing**  
Doplnenie testovania do produkcie – monitorovanie, A/B testy, canary analýzy a chaos engineering už po nasadení, na doplnenie klasických testov.

--

🔹 **DevOps culture (kultúra DevOps)**  
Spoločný prístup tímov vývoja a prevádzky, ktorý zdôrazňuje automatizáciu, spoluprácu a neustále doručovanie (CI/CD). Tester v DevOps tíme participuje na návrhu pipeline, sleduje metriky nasadení a pomáha rýchlo identifikovať regresie.

🔹 **Release management (riadenie vydaní)**  
Proces plánovania, koordinácie a monitorovania nasadení softvéru. Zahŕňa definovanie termínov, zodpovedností, rollback plánov a komunikačné plány – všetko, čo tester potrebuje vedieť, aby pripravil testovacie prostredie a testovacie dáta.

🔹 **Change management (riadenie zmien)**  
Štruktúrovaný proces na schvaľovanie, dokumentovanie a komunikovanie zmien v systéme. Tester sa zúčastňuje posudzovania dopadu zmien, aby vedel, ktoré testy treba spustiť pri každom nasadení.

🔹 **Configuration management (správa konfigurácií)**  
Udržiavanie a verzovanie softvérových a hardvérových nastavení (vstupné parametre, sieťové profily, verzie knižníc). Tester potrebuje konzistentné prostredia, preto sleduje, či sa konfigurácie správne synchronizujú naprieč test, staging a produkciu.

🔹 **Observability (pozorovateľnosť)**  
Schopnosť získať prehľad o vnútri bežiaceho systému pomocou logov, metrík a tracingu. Dobrá observability umožňuje testerovi rýchlo lokalizovať problém a overiť, či testované scenáre spôsobili očakávané udalosti.

🔹 **Monitoring (monitorovanie)**  
Nepretržité sledovanie dostupnosti a výkonnosti systému v testovacích aj produkčných prostrediach. Tester spolupracuje s tímom prevádzky, aby nastavil alarmy na kritické chyby alebo regresné metriky.

🔹 **Incident management (riadenie incidentov)**  
Postup pri riešení neplánovaných výpadkov alebo chýb v produkcii. Tester sa môže zapojiť do post-mortem analýzy, pomáha so zberom dôkazov (test evidence) a overuje, či implementované opravy skutočne chyby odstránili.

🔹 **Service Level Agreement (SLA) (dohody o úrovni služieb)**  
Kontrakt medzi poskytovateľom služby a zákazníkom, ktorý definuje garancie dostupnosti, odozvy a obnovy. Tester musí overiť, či systém dodržiava SLA parametre počas záťažových a stresových testov.

🔹 **Service virtualization (virtualizácia služby)**  
Simulácia správania externých systémov (API, mikroservisov, databáz), ktoré sú ťažko dostupné alebo nákladné. Tester tak môže izolovane overovať logiku aplikácie bez závislosti na reálnych službách.

🔹 **Contract testing (testovanie kontraktov)**  
Overenie, že poskytovateľ a konzument API dodržiavajú dohodnuté špecifikácie (“kontrakty”). Pomáha predchádzať integračným zlyhaniam medzi tímami a zabezpečuje, že zmeny na strane API nepretrhnú dohody.

--

🔹 **Behavior-driven development (BDD)**  
Metodika, ktorá definuje testovacie scenáre v prirodzenom jazyku (napr. Gherkin), spájajúca technickú a netechnickú dokumentáciu. Tester i biznis analytik spolu píšu “user stories” so scenármi formou „Given–When–Then“.

🔹 **Test-driven development (TDD)**  
Prístup, kde sa najprv napíše test, potom ho kód “červený” (zlyhá), následne sa implementuje kód tak, aby test prešiel (“zelený”) a potom sa refaktoruje – princíp Red-Green-Refactor.

🔹 **End-to-end testing**  
Komplexné testovanie celého toku aplikácie od UI až po databázu a späť. Simuluje reálne používateľské scenáre a overuje integritu všetkých vrstiev.

🔹 **API testing**  
Cielené testovanie REST/SOAP či GraphQL rozhraní. Overuje kódy odpovedí, schému dát, čas odozvy a správanie pri chybových vstupoch.

🔹 **UI testing (User Interface testing)**  
Automatizované alebo manuálne overovanie funkčnosti a vzhľadu používateľského rozhrania – tlačidlá, formuláre, navigácia, responzívnosť.

🔹 **Cross-browser testing**  
Testovanie webových aplikácií naprieč rôznymi prehliadačmi (Chrome, Firefox, Safari, Edge) a ich verziami, aby sa zabezpečila konzistentná funkcionalita a vzhľad.

🔹 **Fault injection**  
Cielené vkladanie chýb (sieťové oneskorenie, zlyhanie služby, nedostatok pamäte) do systému, aby sa overila jeho odolnosť a správne ošetrenie výnimiek.

🔹 **Synthetic monitoring**  
Automatizované “robotické” testy, ktoré pravidelne kontrolujú dostupnosť a výkon kľúčových funkcií na produkcii z rôznych geografických lokalít.

🔹 **Mobile testing**  
Testovanie aplikácií na mobilných zariadeniach vrátane simulátorov a reálnych zariadení, so zameraním na rôzne OS (iOS, Android), rozlíšenia a hardvérové špecifiká.

🔹 **Localization testing**  
Overovanie, či je aplikácia správne prispôsobená pre rôzne jazyky a regióny – formáty dátumov/času, preklady, rozloženie textu.

🔹 **Penetration testing**  
Etický “útok” na aplikáciu s cieľom nájsť bezpečnostné dierky (SQL injection, XSS, CSRF, zle nastavené autorizácie) predtým, než zneužijú útočníci.

🔹 **Compliance testing**  
Kontrola, či systém spĺňa regulačné požiadavky (napr. GDPR, PCI DSS, ISO normy), vrátane auditných záznamov a ochrany osobných údajov.

🔹 **A/B testing**  
Porovnanie dvoch verzií funkcie alebo UI prvku (Variant A vs. Variant B) na vzorke používateľov, aby sa vyhodnotilo, ktorá prináša lepší výsledok (konverzie, retention).

🔹 **Test harness**  
Rámec alebo sada nástrojov (skripty, knižnice), ktorá umožňuje spúšťať a reportovať výsledky testov (napr. JUnit, pytest, TestNG).

--