**Óraterv: A szoftvertesztelés hét alapelve**

**Célcsoport:** 12. évfolyam, informatikai ágazat, szoftverfejlesztő és -tesztelő szakma

**Időkeret:** 45 perc

**Tantárgy:** Szoftvertesztelés

**Előzetes ismeretek:** A diákok rendelkeznek alapvető programozási ismeretekkel és tisztában vannak a szoftverfejlesztés alapvető fázisaival (tervezés, kódolás, stb.).

**Az óra menete**

**1. Bevezetés és figyelemfelkeltés (5 perc)**

* **Cél:** A diákok érdeklődésének felkeltése a téma iránt, a tesztelés fontosságának hangsúlyozása.
* **Módszer:** Interaktív kérdések, rövid történet.
* **Tevékenység:**
  + Kezdje az órát egy gondolatébresztő kérdéssel: "Ki találkozott már olyan szoftverrel vagy játékkal, ami lefagyott, hibásan működött, vagy egyszerűen csak idegesítő volt használni?"
  + Rövid beszélgetés a diákok tapasztalatairól.
  + Hozzon egy valós példát egy híres szoftverhibáról (pl. az Ariane 5 hordozórakéta 1996-os felrobbanása egy szoftverhiba miatt, vagy egy ismert játék bugos indulása).
  + Vezesse fel a témát: "Ezek a hibák elkerülhetők lettek volna alapos teszteléssel. A mai órán megismerkedünk a szoftvertesztelés hét alapelvével, amelyek iránytűként szolgálnak a minőségi szoftverek készítéséhez."

**2. A szoftvertesztelés 7 alapelvének bemutatása (30 perc)**

* **Cél:** Az alapelvek megismertetése, megértetése és gyakorlati példákkal való szemléltetése.
* **Módszer:** Előadás, diszkusszió, példaelemzés.
* **Tevékenység:** Az egyes alapelveket egymás után, logikusan felépítve mutassa be. Minden elvnél használjon egy egyszerű, a diákok számára is érthető példát.

**1. Alapelv: A tesztelés a hibák jelenlétét mutatja ki, nem a hiányukat.**

* **Magyarázat:** A tesztelés célja, hogy hibákat találjon. Ha egy teszt nem talál hibát, az nem jelenti azt, hogy a szoftver hibátlan, csupán azt, hogy az adott tesztesetek nem mutattak ki hibát. Soha nem lehetünk 100%-ig biztosak abban, hogy egy szoftver teljesen hibamentes.
* **Analógia:** Olyan, mint a fémkeresőzés. Ha a detektor nem jelez, az nem azt jelenti, hogy az egész tengerparton nincs fém, csak ott nem találtunk, ahol kerestük.
* **Példa:** Egy webáruház bejelentkezési funkcióját teszteljük 10 különböző felhasználónév/jelszó kombinációval. Mind a 10 sikeres. Ez azt jelenti, hogy a bejelentkezés tökéletes? Nem, csak azt, hogy ezzel a 10 esettel működik. Lehet, hogy a 11. eset, például egy speciális karaktert tartalmazó jelszó, már hibát okozna.

**2. Alapelv: A kimerítő (minden lehetőséget lefedő) tesztelés lehetetlen.**

* **Magyarázat:** Egy közepesen bonyolult szoftver esetén is a lehetséges bemeneti kombinációk és végrehajtási útvonalak száma csillagászati. Lehetetlen minden egyes variációt letesztelni.
* **Példa:** Képzeljünk el egy egyszerű szövegbeviteli mezőt, ami 20 karaktert fogad el, és a karakterkészlet 100-féle karakterből áll. A lehetséges kombinációk száma 10020, ami egy felfoghatatlanul nagy szám. Még ha csak a számokat fogadja is el (10 karakter), a kombinációk száma 1020.
* **Következtetés:** Ehelyett a kockázatelemzésre és a prioritások felállítására kell fókuszálni.

**3. Alapelv: A korai tesztelés időt és pénzt takarít meg.**

* **Magyarázat:** Minél később találunk meg egy hibát a fejlesztési ciklusban, annál költségesebb a javítása. A követelmények fázisában talált hiba javítása filléres tétel a már kiadott, ügyfelek által használt szoftverben felfedezett hiba javításához képest.
* **Szemléltetés:** Mutasson egy grafikont, ami bemutatja a hibajavítás költségének exponenciális növekedését a fejlesztési fázisok függvényében (követelmények -> tervezés -> kódolás -> tesztelés -> üzemeltetés).
* **Példa:** Ha már a tervdokumentációban észrevesszük, hogy egy számítási logikát rosszul írtak le, azt egy mondat átírásával lehet javítani. Ha ugyanez a hiba csak a kész szoftverben derül ki, akkor újra kell tervezni, kódolni, tesztelni és telepíteni az egész modult.

**4. Alapelv: A hibák klaszterekbe (csoportokba) tömörülnek.**

* **Magyarázat:** A tapasztalat azt mutatja, hogy a hibák jellemzően nem egyenletesen oszlanak el a szoftverben. A szoftver egy kis részében (a legbonyolultabb modulokban) található a hibák nagy része. Ez a Pareto-elv (80/20 szabály) egyfajta megnyilvánulása.
* **Példa:** Egy vállalatirányítási rendszerben valószínűleg több hiba lesz a bonyolult számlázási és készletkezelési modulban, mint az egyszerű felhasználói profilbeállítások oldalon.
* **Stratégia:** Ha egy területen hibát találunk, érdemes ott alaposabban is körülnézni, mert valószínűleg több is lesz.

**5. Alapelv: A peszticid-paradoxon.**

* **Magyarázat:** Ha ugyanazokat a teszteseteket futtatjuk le újra és újra, egy idő után ezek már nem fognak új hibákat találni. Ahogy a rovarok is rezisztenssé válnak a peszticidekkel szemben, úgy a szoftver is "immunissá" válik az ismétlődő tesztekre.
* **Analógia:** Ha mindig ugyanazon az útvonalon járunk haza, egy idő után már nem veszünk észre új dolgokat. De ha egy másik utcán megyünk, új élmények érhetnek.
* **Megoldás:** A teszteseteket rendszeresen felül kell vizsgálni, frissíteni és új teszteseteket kell írni, hogy a szoftver eddig nem tesztelt részeit is lefedjük.

**6. Alapelv: A tesztelés kontextusfüggő.**

* **Magyarázat:** Nincs egyetlen, univerzálisan legjobb tesztelési stratégia. Másképp tesztelünk egy életmentő orvosi műszert vezérlő szoftvert, mint egy egyszerű mobiljátékot vagy egy céges weboldalt. A tesztelés mélysége, szigora és módszerei az adott szoftver kockázataitól és céljától függnek.
* **Példa:**
  + **Orvosi szoftver:** A megbízhatóság és a biztonság a legfontosabb. Rendkívül szigorú, mindenre kiterjedő tesztelés szükséges.
  + **E-kereskedelmi oldal:** A használhatóság, a teljesítmény és a biztonság kritikus. Fontos, hogy a fizetési folyamat zökkenőmentes legyen.
  + **Mobiljáték:** A szórakoztatás és a felhasználói élmény a lényeg. A tesztelés a játékmenetre, a grafikára és a teljesítményre fókuszálhat.

**7. Alapelv: A "nincs hiba" téveszme.**

* **Magyarázat:** Hiába tesztelünk egy szoftvert hibátlanra, ha az nem felel meg a felhasználói igényeknek és elvárásoknak. Egy hibáktól mentes, de használhatatlan szoftver üzletileg értéktelen.
* **Példa:** Kifejlesztünk egy tökéletesen működő, hibamentes számológép programot, ami csak római számokkal tud számolni. Műszakilag hibátlan, de a legtöbb felhasználó számára használhatatlan, mert nem felel meg a modern elvárásoknak.
* **Konklúzió:** A tesztelésnek nemcsak a hibakeresésre (verifikáció), hanem arra is ki kell terjednie, hogy a szoftver a megfelelő termék-e, és azt csinálja-e, amire a felhasználónak szüksége van (validáció).

**3. Összegzés és lezárás (10 perc)**

* **Cél:** A tanultak rendszerezése, megerősítése, a diákok aktivitásának fenntartása.
* **Módszer:** Interaktív összefoglalás, kérdezz-felelek, házi feladat kiadása.
* **Tevékenység:**
  + **Gyors ismétlés:** Kérje meg a diákokat, hogy saját szavaikkal fogalmazzák meg az egyes alapelveket. Például: "Ki tudná elmondani, mit jelent a peszticid-paradoxon?"
  + **Diszkusszió:** Tegyen fel egy záró kérdést: "A hét alapelv közül szerintetek melyik a legfontosabb egy agilis fejlesztési környezetben? Miért?"
  + **Házi feladat:** Válasszanak egy általuk ismert és használt szoftvert (pl. egy közösségi média app, egy játék, egy szövegszerkesztő). Írjanak le 2-3 olyan lehetséges hibát, amivel találkozhatnak benne, és párosítsák a hibákat a ma tanult alapelvek valamelyikével! (Például: "A játék egy ritka tárgykombinációnál mindig lefagy. -> Hibák klaszterekbe tömörülnek.")
  + **Előretekintés:** Jelezze a következő óra témáját, ami lehet például a tesztelési szintek (egységteszt, integrációs teszt, rendszer teszt) vagy a tesztelési típusok (funkcionális vs. nem-funkcionális).