Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Nagy Dániel Dávid

FÉLSTRUKTURÁLTADAT-RÖGZÍTŐ FELÜLET FELHASZNÁLÓ-KÖZPONTÚ TERVEZÉSE

KONZULENS

Kővári Bence

BUDAPEST, 2011

Tartalomjegyzék

Összefoglaló

Képzett kutatók drága segédeszközéből a számítógép napjainkra az otthonok és a munkahelyek szerves részévé vált. A felhasználók zöme számítástechnikailag képzetlen és esetenként teljesen tapasztalatlan is: nem ismeri és nem is célja megismerni számítógépe működését, inkább csak hasznos segédeszközként tekint rá feladatai elvégzésében. Egy szoftver, ha nem rendelkezik egyszerű, könnyen használható, intuitív és esztétikus kezelőfelülettel, gyakorlatilag értéktelen a felhasználók számára – bármilyen jól megtervezett, precízen implementált termékről legyen is szó.

A dolgozat áttekinti a grafikus felhasználói felületek kifejlődését övező kutatások és termékek fontosabb tanulságait, és az akkori kutatókban megfogalmazódó alapelveket, amelyek mára gyakorlatilag kikerülhetetlen elvárássá váltak a végfelhasználói szoftverekkel szemben. Bemutatásra kerülnek a felhasználói élmény ellenőrzését és javítását célzó modern tesztelési metódusok alaptípusai, és egy iteratív fejlesztési modell, ami a használhatóság tesztek és a fokozatos javítás folyamatos ismétlésével költséghatékonyan biztosíthatja akár egy kevésbé triviális, újszerű megoldásokat alkalmazó felület intuitív és kényelmesen használható kialakítását, a leendő végfelhasználók előképzettségéhez és tudásszintjéhez igazítva azt.

A bemutatott módszer lehetőséget nyújtott egy újszerű, általános megoldással máig nem rendelkező, de egyre nagyobb jelentőséggel bíró probléma, a félstrukturált adatok kezelését biztosító adatrögzítő felület kialakításának felhasználóbarát megvalósítására. A dolgozat esettanulmányként egy átfogó színházikellék-nyilvántartó rendszer beviteli felületének megtervezését, és felhasználói visszajelzéseken alapuló továbbfejlesztését járja körül, különös hangsúlyt fektetve az újszerű vizuális és interaktív elemekre, majd ezek kiértékelésére valósághű környezetben. A dolgozat végén mind egy, a tesztelés tapasztalatai alapján módosított változat, mind néhány hosszabb távú, átfogóbb továbbfejlesztési lehetőség is bemutatásra kerül.

Az alkalmazott felhasználó-központú, iteratív fejlesztési modell lehetővé tette egy félstrukturált adatok kezelésére szolgáló, felhasználóbarát felület kialakítását, ellenőrzését, és folyamatos javítását. A végzett felhasználói tesztek tanulsága szerint a bemutatott koncepció könnyen elsajátítható, kényelmes és hatékony adatrögzítést tett lehetővé az azt kipróbáló felhasználók körében, az eredmények ugyanakkor iránymutatást is adtak mind rövid-, mind hosszabb távú fejlesztések elvégzéséhez.

Abstract

From an expensive aid of qualified researchers, computers became part and parcel of homes and workstations recently. Most users are inexperienced and unqualified in terms of informatics: they do not know or want to know how their computer works, but regard it as a useful tool to aid the completion of their tasks. Without a simple, easy to use, intuitive and aesthetic user interface (UI), software is practically worthless for most users, no matter how well-designed, precisely implemented product it really is.

This paper explains some of the lessons learnt from researches and products from the early days of graphical user interfaces, and the basic principles proposed by researchers of that time, which became practically unavoidable requirements of end user software since then. The basic types of modern usability testing methods aiming at checking and improving the user experience, get introduced then, along with an iterative development model, which can cost-effectively allow for designing an intuitive, easy to use interface with non-trivial, novel elements, by repeatedly conducting usability tests and implementing improvements continuously based on their results.

The presented method allowed for creating a user-friendly approach to a novel, but increasingly important problem, that lacks a general solution thus far, which is providing a flexible data entry form to handle semi-structured data. As a case study, the paper describes the designing of the data entry UI of a comprehensive theater prop management system, as well as improving it based on user feedback, with special emphasis on novel visual and interactive elements, and their evaluation in a realistic environment. By the end of this paper, a modified version of the form – with improvements implemented based on usability test results – gets introduced as well, along with a few long-term, more comprehensive enhancement opportunities.

The applied user-centered, iterative development model made it possible to design, evaluate and constantly improve a user-friendly interface for managing semi-structured data. According to the results of the conducted usability tests, the presented concept provided an easy to learn, comfortable and efficient way to enter semi-structured data, among participating users. Possible improvements were also suggested by the findings, both short-term and long-term ones.

1 Bevezetés

1.1 A felhasználói felület jelentősége és kihívásai

Bár a számítógép alapötletének születése egészen a XIX. századig nyúlik vissza, általánosan elterjedtté, mindenki számára elérhetővé válásához közel száz évnek kellett eltelnie. Több generáción átívelő fejlesztésre volt szükség, a gyártástechnológia és az alapvető működés finomhangolásában egyaránt. Az 1980-1990 közötti időszakhoz köthető az ún. személyi számítógép (Personal Computer, PC) – mint fogalom, s mint termék – megjelenése; ez volt az első sikeres próbálkozás arra, hogy a korábban súlyos beruházásokat kitevő hardver a kutatólaboratóriumokból bárkinek az otthonába eljuthasson, segítse feladatai elvégzésében [1]. A célközönség drasztikus mértékű kiterjedése új elvárásokat fogalmazott meg a számítógéppel szemben. A PC-jüket munkájuk hasznos segédeszközének tekintő *rendszeres felhasználóktól* egészen ún. *eseti felhasználókig* – akik interakciója jellemzően kioszkok, például könyvtári nyilvántartások, utastájékoztatók stb. kezelésében kimerül – bővült a korábban csak részletes és mélyreható szaktudással rendelkezőekre kiterjedő skála [2]. Az újonnan megjelenő csoportok legfeljebb csak minimális – nemritkán semmilyen – számítógépes tudás megszerzésére hajlandók, a kényelmes, magától értetődő, egyszerű kezelést éppúgy elvárják digitális munkaeszközeiktől, mint a korábban megszokott, őket körülvevő fizikai tárgyaktól. Bár az elvárás jogosnak és kézenfekvőnek tűnik, az ipar kezében máig nincs egyértelmű megoldás az új követelmények kielégítésére. Egy mérnöki szempontból jól megtervezett, precízen implementált szoftver, ha nem rendelkezik egy vizuálisan és használhatóság tekintetében egyaránt kielégítő kezelőfelülettel, felhasználói – és így üzleti – szempontból gyakorlatilag értéktelen. A Microsoft Office 2003 irodai szoftvercsomag esete jól példázza a jelenséget. Bár az eladásokat illetően alapvetően egy sikeres termékről van szó, a megjelenést követő közvélemény-kutatások egyértelműen azt mutatták, messze alulmúlta azt a potenciált, amit a programozói oldalról belefektetett munka jelenthetett volna számára. A következő verzióhoz javasolt funkciók, szolgáltatások listájának kitöltésekor ugyanis igen gyakran olyan elemeket neveztek meg a felhasználók, amelyek az aktuális, általuk megvásárolt kiadásból sem hiányoztak – egyértelműen rávilágítva ezzel az akkori felhasználói felület kudarcára. Nem pusztán az elvégzendő feladat megoldásának

segítésében múlta alul az elvárásokat, egyenesen azt a képzetet keltette, hogy az adott feladat megoldására nincs is mód a termék használatával. Mind a befektetett programozói munka kárba vesztét, mind a potenciális piaci visszaesést szem előtt tartva döntött ezért úgy az Office fejlesztői csapata, hogy a következő, 2007-es kiadásukban a felhasználói felület újragondolására kell helyezniük a hangsúlyt [3].

Ahogy az ipari tervezés az 1920-as években, a szoftvertervezés is egy átmeneti időszakban jár [4]. Napjainkban ugyan már természetes, hogy a fizikai formát öltő termékek nem pusztán a technikai, hanem az ergonómiai igényeket is ugyanúgy kielégítik, ez korántsem volt mindig így. Noha a korszerű formatervezés egyik zászlóshajója az autóipar, az eredeti Ford T-modell vásárlói nem pusztán a kocsi színét nem választhatták meg, számtalan használati kényelmetlenséggel is szembesülniük kellett. Csak később vált nyilvánvalóvá, hogy a mérnöki végzettségű szakemberek mellett termék- és formatervezőkre is egyaránt szükség van a produktum kialakítása során. A XX. század elején azonban ez utóbbi szakmák elsajátítása nem pusztán a megfelelő képzések hiánya miatt ütközött akadályokba, létjogosultságuk is időről időre megkérdőjeleződött, azon illúzió következtében, hogy az ő közreműködésük nélkül is *elég jó* termékek készülnek – a felhasználói tapasztalatok, reakciók mélyreható kutatására volt szükség a cáfolat általános elfogadtatásához.

A szoftverek kezelhetőségével kapcsolatos problémák többsége is egy teljesen hasonló anomáliára vezethető vissza: nevezetesen, hogy azok felhasználói felületét igen gyakran – még manapság is – mérnökök, programozók készítik (a konkrét funkcionalitás megtervezése, implementálása mellett), a megfelelő pszichológiai, ergonómiai stb. képzettség nélkül. Mitchell Kapor fogalmazta meg először – 1990-ben – a szoftvertervező, mint a szükséges mérnöki és humán tudományok határán mozgó, önálló tudományág kialakításának szükségét [5]. Napjainkban, bár az ehhez szükséges képzések egyre nagyobb számban állnak rendelkezésre, több helyen továbbra is problémát jelent a menedzsment és a programozók munkájának összehangolása, a megfelelő szakemberek híján [4]. Gyakran lépnek fel prioritásbeli problémák is: még ha részt is vesznek hozzáértő grafikus tervezők az alkalmazás fejlesztésében, a vonzó megjelenés fontossága sokszor túlértékeltté válik a használhatóság jelentőségével szemben – ilyenkor a döntések esztétikai és nem ergonómiai megfontolások alapján születnek, amelyeket egyéni, személyes preferenciák, és nem a leendő felhasználók valós igényei vezérelnek [10].

Bár a kapcsolódó ergonómiai kérdések kutatása több évtizedes múltra visszatekintő, manapság is releváns és elismert eredményekkel büszkélkedhet, és jól bevált, bevett gyakorlatok széles skálája segíti a grafikus felületek felhasználóbarát kialakítását, egy szoftver tényleges használhatósága – beleértve a valódi célközönség igényeinek való megfelelést, a kényelmes és hatékony kezelhetőséget és a könnyű elsajátíthatóságot egyaránt – csak a megfelelő tesztek végrehajtása után, a felhasználói visszajelzések kiértékelésével ítélhető meg.

Ennek ellenére a felhasználói tesztek elvégzése a legtöbb szoftver életciklusából kimarad – még ha le is zajlik, akkor is többnyire a fejlesztés végén, a már teljesen elkészült alkalmazás kipróbálásával történik, amikor már szinte lehetetlen a felfedett hiányosságok orvoslása, a javítási lehetőségek implementálása. A felület ugyanis ekkorra már túlzottan összetetté válik ahhoz, hogy más, jól működő funkciók sérülése nélkül érdemben módosítható legyen, az idő- és anyagi keret pedig ebben a szakaszban már általában nem elegendő nagyméretű átalakítások kellően alapos véghezviteléhez [13].

Különösen fontos a tesztelés korai megkezdése, ha a felület újszerű, nem triviális működésű elemeket tartalmaz. Például a legtöbb űrlapalapú alkalmazás teljes egészében felépíthető jól ismert, bevált, hagyományos beviteli vezérlők – mint amilyen a szövegdoboz, a legördülő lista vagy a jelölőnégyzet – megfelelő elrendezésével; ha követjük a már kialakult irányelveket és egyszerű ergonómiai ökölszabályokat – mint amilyen a TAB billentyű helyes kezelése, vagy a beviteli mezők címkéinek jól érthető elhelyezése –, könnyen alakíthatunk ki hatékony és felhasználóbarát felületet. E megközelítés azonban az űrlapon keresztül megadandó adatok szigorú tipizálását, strukturálását igényli – nincs lehetőség a típusok módosítására a felület hasonló súlyú átalakítása nélkül. Bár félstrukturált adatok tárolására és kezelésére már számos megoldás született – mint például az eXtensible Markup Language (XML) formátum, és a feldolgozására szolgáló Simple API for XML (SAX) –, grafikus felületeken való szerepeltetésük, hatékony és felhasználóbarát rögzítésük továbbra sem megoldott – nincs általános, alaposan tesztelt vezérlőkészlet ilyen rugalmas felépítésű információ megadására.

1.2 A dolgozat felépítése

A következő szakaszban áttekintem a korai grafikus felhasználói felületek kialakításának lényegesebb alapelveit, amelyek gyakorlatilag máig általános elvárásoknak tekinthetőek a kliensalkalmazásokkal szemben. Ezt követően ismertetem a felhasználói tesztek alaptípusait, melyek a jól használható, pozitív élményt nyújtó felületek elkészítésének és kiértékelésének legfontosabb eszközei közé tartoznak, mégis gyakran kimaradnak a mai szoftverprojektekből. Bemutatok egy idő- és költséghatékony megközelítést e tesztek fejlesztési ciklusba történő integrálására, majd esettanulmányként, a 3. fejezetben bevezetem egy színházi kelléknyilvántartó rendszer fejlesztési folyamatát, ami a leltáralkalmazások egy speciális eseteként újszerű kihívást állít a felülettervezés elé: a félstrukturált adatok kényelmes és hatékony rögzítési módjának megtalálását. A fejezetben ismertetem az ebből fakadó szokatlan adatszerkezeteket és a felhasználói interfésszel szemben támasztott elvárásokat, áttekintem a teljes rendszer architektúráját, majd részletesen is kitérek az általam megvalósított adatrögzítő felület felépítésére, működésére. Végül bemutatom az alkalmazott tesztelési folyamatot, amelyet megoldásom ellenőrzésére használtam, és az eredményeket is kiértékelem, részletezve mind a rövid távú, már implementált módosítási javaslatokat, mind a hosszabb távú, koncepcionális kérdéseket is érintő továbbfejlesztési lehetőségeket.

2 A felhasználói visszajelzés szerepe az iteratív fejlesztési ciklus folyamán

2.1 A felhasználói felületekkel szemben támasztott elvárások

A személyi számítógépek megjelenése előtt a tipikus felhasználói felület karakter alapú volt, parancsok memorizálását és hibátlan megadását, jó gépelési képességeket, és a rendszer működésének mélyebb ismeretét igényelte [2]. Az első grafikus felhasználói felület (Graphical User Interface, GUI) kialakítására irányuló kutatásoknak a Xerox Palo Alto Research Center (PARC) adott otthont [6]. Bár menedzsment- és teljesítményproblémák miatt a PARC első személyi számítógépe, a Star nem váltotta be maradéktalanul a hozzá fűzött reményeket, a tervezői által megfogalmazott alapelvek máig visszaköszönnek a korszerű PC-k kezelőfelületén [7]. Néhány fontosabb a máig is érvényesek közül:

Közvetlen kezelés: Az elvégezhető műveletek nem szöveges parancsokkal, hanem a képernyőn vizuálisan megjelenő, valós tárgyakra emlékeztető objektumokon keresztül érhetőek el. A Star az ún. asztal metaforát vezette be ezen objektumok definiálásakor: a felhasználó dokumentumokat, nyomtatókat, mappákat, aktaszekrényeket, ki- és bemenő postafiókokat stb. választhat ki, egyszerűen rájuk mutatva[1], a számítógép utasításához[2].

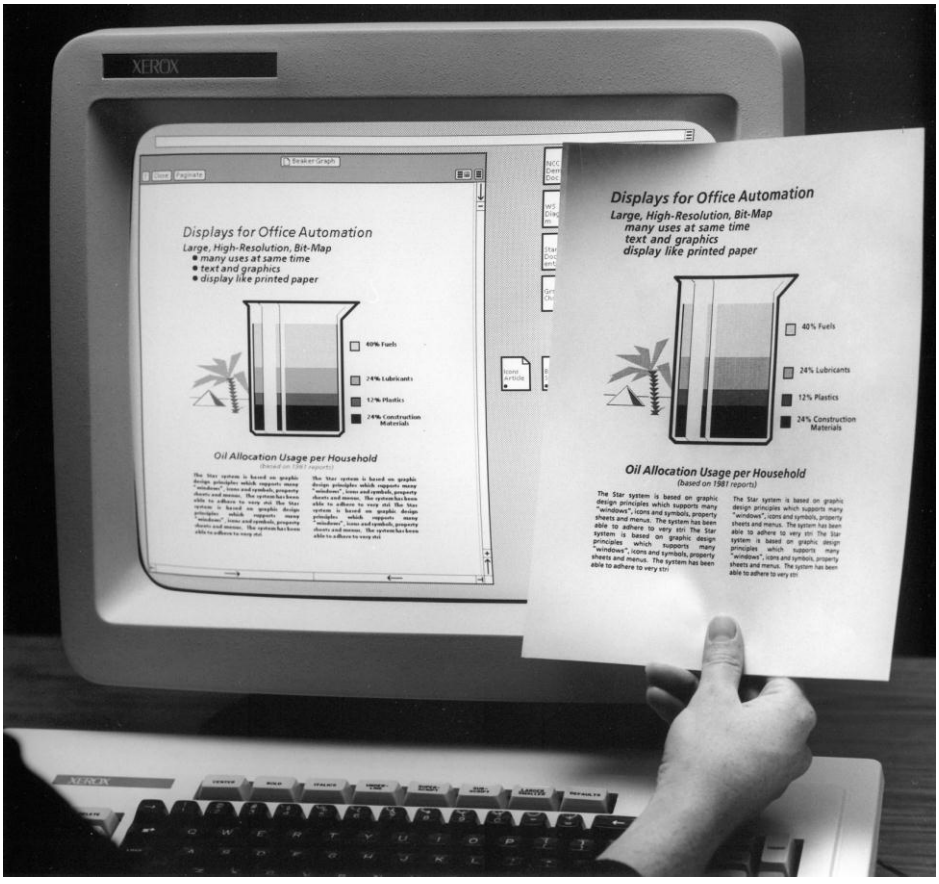
Azt kapod, amit látsz (What You See Is What You Get, WYSIWYG): Korábban a bonyolultabb grafikus kimenettel rendelkező programok – így például rajzeszközök, összetettebb, különféle betűtípusokat támogató szövegszerkesztők stb. – kezelésekor a felhasználó szövegesen, valamiféle speciális szintaxissal kellett, hogy dolgozzon; a végeredmény vizuális megjelenésével csak később, a kód szoftver általi értelmezése után találkozott, ami nagymértékben nehezítette munkáját, és specifikus tudást igényelt. A grafikus kimenet szerkesztés közben történő megjelenítését, közvetlen manipulálhatóságát – amely manapság az említett programkategóriákban gyakorlatilag

[1] Tipikusan egérrel.

[2] Dokumentum nyomtatásához például a dokumentum és a nyomtató ikonját kell használni.

alapkövetelmény – a Xerox Star vezette be. (**Hiba! A hivatkozási forrás nem található.** [8].)

Parancsok egységessége (konzisztenciája): Az általános parancsok használata – így például a másolás, törlés, áthelyezés, megnyitás stb. – a rendszer egészére nézve egységes: az egyik program használatakor megtanult művelet – ha értelmezhető – egy másik program futása során is ugyanúgy használható (például ugyanúgy másolható egy-egy szó egy szövegszerkesztőben, mint a dokumentum mappáról mappára). Ez nagyban lerövidíti a tanulási ciklust, egyszerűsíti a használatot, és egy egységes, átgondolt rendszer érzetét kelti. Megjegyzendő, hogy a Xerox azáltal tudta ezt az egységességet biztosítani, hogy a Staron elérhető valamennyi alkalmazást ők készítették, a szoftvercsomag nem volt bővíthető külső fejlesztők által – ami nem kis hátulütője volt a terméknek a konkurens – bár jóval szerényebb felhasználói felülettel rendelkező – megoldásokhoz képest.



1. ábra: A WYSIWYG elv a gyakorlatban [8].

A grafikus felhasználó felület ötletét elsőként az Apple vitte tovább. A PARC eredményeit jelentősen újragondolva, erős menedzsment- és marketingtámogatással, 1984-ben megjelenő Macintosh számítógép korának egyik legsikeresebb

számítástechnikai árucikke lett [6]. Sikerében nem kis szerepe volt annak, hogy a Starban megjelenő egységesség elvet az Apple – a Xeroxszal ellentétben – meg tudta valósítani anélkül, hogy a külső gyártókat kizárta volna a rendszerre való szoftverfejlesztésből. Így megőrizte a bővíthetőséget, ami a konkurens – grafikus felülettel még nem rendelkező – megoldásokat népszerűvé tette.

Az Apple módszere a konzisztencia elérésére egy tudatos kampány véghezvitele volt, melynek keretében számos szakemberük foglalkozott a külsős cégek felkeresésével és meggyőzésével, hogy alkalmazásaik interfészét *Macintosh módon* alakítsák ki (akkor is, ha személyes meggyőződésük alapján a saját megoldásukat találnák preferálni), a beépített programok által bevezetett felületelemek és viselkedés követésével [7].[3] A javaslatok követése a fejlesztők számára is kézzelfogható előnyökkel bírt: egy Macintosh felhasználó számára szinte bármilyen új program az ismerősség érzését sugározta, kényelmesen és egyszerűen használatba vehetővé téve azt – pontosan azért, mert a korábban már megtanult működés alkalmazható volt az új szoftverre is. Az egyszerű kezelhetőség, természetesen, a termék népszerűségében, így az eladásokban is megmutatkozik.

A propagált irányelvek – a széleskörű elterjedés érdekében – később[4] nyomtatásban is megjelentek. Ezek némelyike a PARC kutatói által megfogalmazottak újragondolása *(metaforák, közvetlen kezelés, rámutatás, WYSIWYG, konzisztencia),* de számos újabb, szintén általános érvényűnek mondható ökölszabály is megfogalmazásra került [9]:

- *A felhasználó irányít*: A műveletek elsődleges kezdeményezője a felhasználó kell, hogy legyen. Nem szerencsés, ha a szoftver végez el helyette mindent – az általa jónak vélt módon –, hiszen így nem csak az eredmény múlhatja alul az ő eredeti elképzeléseit, a rendszer használatát is nehezebben fogja megtanulni később. Még akkor sem szabad kivenni az irányítást a felhasználó kezéből, ha az általa választott opció potenciális veszélyt jelenthet az adataira, vagy akár a számítógépre nézvést – természetesen

[3] Így például ha egy ikon már valamilyen létező jelentéssel bír a rendszerben, egy külső szoftvernek nem szabad más jelentéssel felruháznia azt; hasonlóan nem előnyös új ikonokat bevezetni olyasmikhez, amelyeknek már létezik beépített vizuális megjelenése.

[4] Az 1980-as évek végén.

célszerű egy figyelmeztetést megjeleníteni, de lehetőséget kell biztosítani a művelet folytatására.

- *Visszajelzések és dialógusok*: A felhasználónak mindig, minden esetben azonnali visszajelzésre van szüksége. Tudnia kell, hogy a kérése sikeresen végrehajtódott-e; ha várnia kell, akkor mire, és meddig; ha nem sikerült a művelet, akkor miért nem, illetve mit tehet a probléma elhárítása érdekében. Mindez természetesen nem szabad, hogy összezavarja a felhasználót, így például, bár a hibaüzenetekben célszerű minél több információt megjeleníteni, a hibakódok tipikusan használhatatlanok az átlagember számára, így érdemes elrejteni őket.
- *Megbocsátás*: A felhasználói műveletek visszavonhatóak kell, legyenek – ezzel is bátorítva a lehetőségek kipróbálását, a kísérletezgetés jellegű tanulást. Amennyiben a visszafordíthatóság valamely funkciónál nem valósítható meg, mindenképpen figyelmeztetni kell a felhasználót a kérés végrehajtása előtt, hogy adott esetben meggondolhassa magát.
- *A stabilitás érzete*: Mivel a számítógép a legtöbb felhasználó számára túl komplexnek ígérkezhet elsőre, stabil referenciapontokra van szükségük, hogy biztonságosnak érezzék a használatát. A menüpontokat mindig ugyanott kell elhelyezni (ha éppen nem érhető el valamelyik, akkor sem szabad eltűntetni, csak elhalványítani, a hiábavaló keresést elkerülendő), és kerülni kell az egyéb, váratlan módosulásokat is – például lehetőség szerint sose változzon az ablakok mérete, hacsak nem ez volt a felhasználó kifejezett szándéka.
- *Esztétikai egységesség*: Az alkalmazás grafikus felületének illeszkednie kell a Macintosh rendszer egészéhez – a szokásos vezérlőelemek, ikonok, viselkedés stb. alkalmazásával. Az ismerősség érzése mellett fontos továbbá a jó megjelenés – elvégre a felhasználó sokszor, sokáig kell, nézze majd a GUI-t képernyőn – és az ergonómiai alapelvek betartása is. Egyszerű, logikus elrendezésre van szükség – a profi megjelenés a profi, megbízható működés érzetét kelti. Érdemes kiemelni, hogy az Apple már ekkor propagálta szakképzett grafikus designerek alkalmazását a felület megjelenítésének kialakításához – ez az elem sok szoftverfejlesztő csapat munkafolyamatában máig nem jelenik meg.

- *Módnélküliség*: Bár a korábbi alkalmazások funkcióinak elterjedt szervezési elve volt a különböző üzemmódok alkalmazása, használatuk a Macintosh platformon kifejezetten ellenjavallttá vált, mivel – az egyes lehetőségek üzemmódfüggő megjelenítése miatt – számos egyéb alapelvet, helyes gyakorlatot sért. Például rontja a stabilitás érzetét, és a felfedezés alapú tanulást is igencsak megnehezíti. Természetesen ez is csak ökölszabály, maga az útmutató is számos kivételt megemlít, amelyek esetén figyelmen kívül hagyható (vagy hagyandó) a módnélküliség elve.

A kézikönyv ezen túlmenően tervezési, előkészítési, tesztelési (ideértve a korábban említett, felhasználók megelégedését kutató használhatóság teszteket) és implementációs tanácsokat is ad, korának egyik legteljesebb felhasználóifelületelem-referenciájával – melyben a ma ismert vezérlők (így például a gomb, a legördülő lista vagy éppen a dialógusablak) gyakorlatilag mindegyike megtalálható, részletes felhasználási iránymutatással. E felületelemek általánossága a Macintosh-ét követő grafikus interfészek használhatóságának is egyik alappillérévé vált.

2.2 Kvalitatív és kvantitatív tesztelés

Bár a korábbiakban említett alapelvek és iránymutatások megbízható támpontot jelentenek az alkalmazás felületének kialakításakor, a felhasználói visszajelzések gyűjtése és feldolgozása egyetlen felhasználóbarát szoftver életciklusából sem hiányozhat. Fontossága és értéke ellenére azonban a felhasználói tesztelés jellemzően nem része a fejlesztési folyamatnak – leginkább a végrehajtásához szükséges idő, pénz és szaktudás hiányára hivatkozva marad ki az ütemtervből [10]. Bár évtizedekkel korábban a visszajelzések gyűjtésének e formája valóban drága és időigényes volt, eredményeinek kiértékelése pedig speciális szakértelmet igényelt, 1989-ben megjelent publikációjában Jakob Nielsen jelentősen leegyszerűsítette a korábban egyeduralkodó módszertant [11]. Az általa „diszkont használhatósági tesztnek” nevezett folyamat lényegesen kevesebb résztvevőt igényel, és könnyen – nehezen megszerezhető szaktudás nélkül – értelmezhető eredményekkel szolgál, így lényegében bármely szoftverfejlesztési projekt költségvetésébe beilleszthető. Bár a tesztelés e formájának is megvannak a költségei, hiánya – hosszabb távon – jóval nagyobb veszteségekkel járhat a várhatóan szerényebb felhasználói élmény következtében, ami terméktámogatás-hívások, megjelenés után kifejlesztendő szervizcsomagok és egyéb utólagos költségek formájában jelentkezik [12].

Hagyományosan a felhasználói tesztek a tudományos kutatásokhoz hasonlítottak. A fő céljuk valamilyen állítás bebizonyítása volt (pl. a felhasználói felület „A” változata jobb, mint a „B” változat, valamilyen számszerűsíthető, pontosan meghatározott szempont szerint, mint például egy adott feladat végrehajtásához szükséges idő), emiatt szigorúan követendő, jól definiált folyamata volt végrehajtásuknak, rengeteg résztvevőt igényeltek – a statisztikailag szignifikáns eredmények biztosítása végett –, és egyedi szaktudásra volt szükség a megfelelő metrikák megválasztása, és a mérések félreértelmezésének elkerülése érdekében [13]. További feltételük volt a kérdéses alternatívák közötti eltérés elkülönítése – vagyis a felület minden más elemének változatlanul hagyása –, ami nagymértékben megnehezítette a felhasználói interfész alapos vizsgálatát, inkább egy-egy kiemelt aspektusára vonatkozólag lehetett következtetéseket levonni. Vannak természetesen helyzetek, amelyek pontosan ilyen jellegű összehasonlító elemzést igényelnek, a legtöbb fejlesztési projekt költségvetésébe azonban nem fér bele egy ilyen alaposságú kutatás, és ha mégis, sem feltétlenül sem képes az igazán komoly problémákra hatékonyan rávilágítani [14].

A Nielsen javasolta metódus inkább kvalitatív visszajelzésekre hagyatkozik, mint nyers számadatokra. A résztvevő felhasználót gondolatai hangos kimondására kérik – például ha valami nem úgy történt, ahogy várta, miután kattintott, vagy min gondolkozik, amikor két menüpont között vacillál stb. – miközben a vizsgált szoftvert használja (tipikusan a tesztvezető által előre kitűzött feladatok végrehajtására), valamint a képernyőről videofelvétel is készül, hogy megjegyzései később is kontextusba helyezhetőek legyenek. Ez a megközelítés kevés résztvevővel is releváns eredményekkel szolgál: ha a felhasználók közül akár csak egy is elakad valamelyik feladat megoldása közben, az kétségkívül használhatósági problémát jelent.[5] A hangos gondolkodásnak köszönhetően pedig nem pusztán a probléma ténye, de annak oka is felfedhető – például ha tudjuk, hogy az érintett résztvevő hogyan próbálta a feladatot megoldani, és ez mely pontokon tért el az alkalmazás tényleges viselkedésétől, jobban behatárolható, a felület mely részei bizonyultak félrevezetőnek, vagy működnek kényelmetlenül. A felhasználó így megfogalmazódó, máskor egyébként kimondatlan elvárásai ezen felül a megoldásra vonatkozólag is hasznos ötleteket adhatnak [13].

[5] A teszt résztvevői ráadásul jellemzően minden tőlük telhetőt megtesznek, hogy megoldják a feladatot. A valós felhasználók jóval hamarabb feladják, ha az alkalmazással nehézségeik adódnak.

2.3 A résztvevők számának meghatározása

Ahogy az előző szakaszban bemutatásra került, kvalitatív teszteket már akár egyetlen résztvevővel is érdemes lehet végrehajtani, hiszen a legsúlyosabb problémák jó eséllyel még ebben a helyzetben is kiderülnek – szemben a tesztelés teljes kihagyásával a fejlesztési folyamatból, ami természetesen semmilyen visszajelzéssel nem szolgál. Nielsen rávilágít azonban, hogy ötnél több felhasználóra nincs is szükség – további résztvevők bevonásával ugyanis jellemzően a már korábban megfigyelt problémákat fogjuk észlelni újra és újra, és csak nagyon kevés új információhoz juthatunk (2. ábra) [15].

2. ábra: A résztvevők számának növelésével egyre kevesebb új problémát figyelhetünk meg [15].

Mindez természetesen nem jelenti azt, hogy egyetlen 5 résztvevős tesztelési ciklus fel fogja fedni a felület használhatósági problémáinak több, mint 75%-át – csupán azon problémák 75%-áról van szó, amelyek az adott tesztelési ciklusban egyáltalán megfigyelhetőek. A többi rejtve marad mind a felhasználók, mind a tesztvezető előtt, mivel súlyosabb problémák vagy hiányzó funkciók következtében az alkalmazás érintett képernyői elő sem kerülnek a teszt során [10].

A kvalitatív tesztek hatékonysága tehát leginkább a tesztelési ciklusok, és nem az egyes ciklusok résztvevőinek számán múlik. A visszajelzések gyűjtését integrálni kell a fejlesztési folyamatba: az egyes tesztelési ciklusok között ki kell javítani a legsúlyosabb hibákat, hogy a korábban rejtett problémák felszínre kerülhessenek. További előnye ennek a megközelítésnek, hogy a későbbi tesztek a korábbiak során

felfedett hibák javításait és az egyéb finomhangolásokat is ellenőrizni tudják: hogy ténylegesen megoldást jelentenek-e az eredeti problémára, és nem vezettek-e be esetleg újakat [16]. Az utóbbi különösen fontossá válik a megjelenés közeledtével: míg a fejlesztés korai szakaszaiban előnyösebbnek bizonyulhat a felület teljes újratervezése, mint minden egyes hibát külön-külön javítani – az alapvető felépítés megtartása mellett –, egy ilyen méretű módosítás olyan részeit is elronthatja a felületnek, amelyekkel korábban nem voltak nehézségeik a felhasználóknak [10] – a fejlesztési ütemtervbe pedig e hibák megtalálása és javítása is bele kell, hogy férjen.

A bemutatott szempontokat figyelembe véve, a kvalitatív felhasználó tesztelést iteratív módon kell alkalmazni: már a fejlesztési ciklus elején meg kell kezdeni az alkalmazás egyes részeinek tesztelését, amit újabb és újabb tesztekkel kell kiegészíteni, ahogy az új funkciók vagy a korábban talált hibák javításai implementálásra kerülnek.

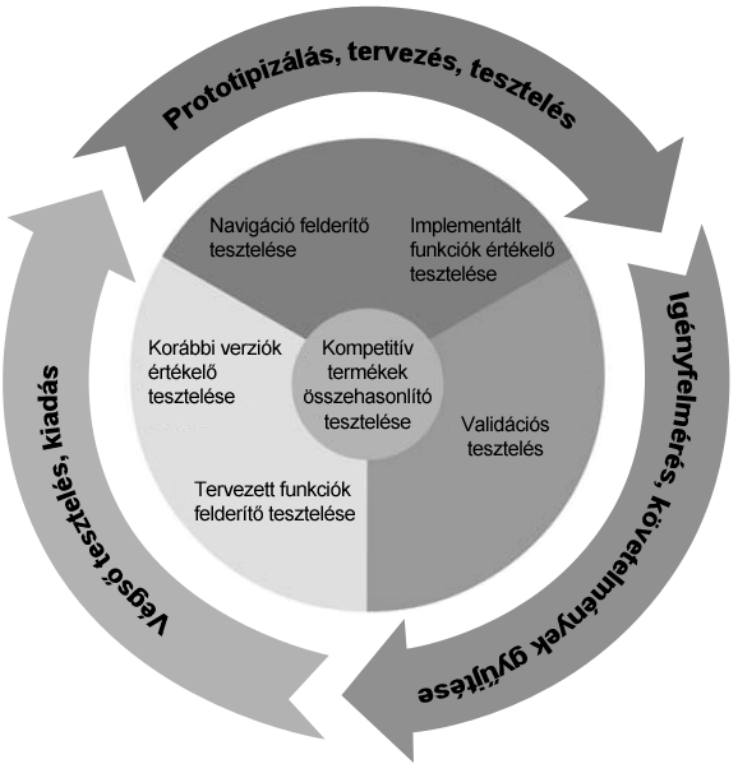
2.4 A tesztelés lehetőségei és céljai a fejlesztési folyamat egyes lépései során

Tesztelés szempontjából az alkalmazás egy verziójának életciklusa az alábbi három fő fázisra bontható (3. ábra) [14]:

Igényfelmérés, követelmények gyűjtése: A tesztek alanya ebben a fázisban általában a termék előző verziója (amennyiben van), illetve hasonló, konkurens termékek lehetnek. A feltárt hiányosságok jó alapot adhatnak az alapvető koncepciók újragondolásához, új funkciók felderítéséhez, a kvalitatív adatok gyűjtésével pedig konkrét mérce állítható be az elkészítendő alkalmazás számára. Összetettebb alkalmazás esetén a navigáció és/vagy menürendszer korai szerkezeti vázlatát is érdemes lehet tesztelni.

Prototipizálás, tervezés, tesztelés: Maga a fő fejlesztési folyamat, melyet a tesztelés mind a korai prototípusok értékelésénél, mind fejlesztés közben, a végleges felület kialakítása során átsző, a korábbiakban bemutatott iteratív modell szerint.

Végső tesztelés, kiadás: Az alkalmazás átfogó ellenőrzése, ahol már nem csak a különálló funkciók felhasználói élménye, hanem azok elrendezése, integráltsága is ellenőrzésre kerül. Az ilyenkor gyűjtött kvantitatív adatok számszerűsítve is ellenőrizhetik az elkészült termék előző verzióhoz vagy konkurens szoftverekhez viszonyított használhatóságát, illetve alapot szolgálhatnak a jövőben elkészítendő verziófrissítések megítéléséhez.



3. ábra: Az alkalmazás életciklusának egyes fázisaiban alkalmazható felhasználóiteszt-típusok [14].

A követelménygyűjtés kései, illetve a tervezés korai szakaszaiban az ún. *felderítő* tesztek dominálnak. Ezek célja az alkalmazás alapvető felépítésének – valamint összetettebb esetben a navigációnak – közös végigjárása, kipróbálása potenciális felhasználókkal, akár hiányosságok vagy lehetséges kényelmi szolgáltatások felfedezése, akár alternatívák informális összehasonlítása végett. A metódus kulcsa a tesztek minél korábbi elvégzése, kifejezetten azt megelőzően, hogy az egyes megjelenésbeli, szerkezeti döntések igazán kiforrottá válnának. Ennek köszönhetően elkerülhető, hogy a fejlesztés már korán – koncepcionális szinten – téves irányba induljon, mivel a korai elképzelések ilyenkor még nagymértékben áttervezhetőek – súlyos esetben akár el is dobhatóak – jelentős extra költségek vagy időveszteség jelentkezése nélkül; később ugyanezen hibák kezelése általában már csak kisebb átalakítások formájában tud megvalósulni, a valódi probléma teljes megszűntetése nélkül, szerényebb felhasználói élményt eredményezve.

A második fázisban az *értékelő* tesztek dominálnak, melyek tárgya az alkalmazás már implementált funkcióinak értékelése; a résztvevő felhasználók már nem pusztán navigálnak az alkalmazásban, hanem valósághű, konkrét feladatokat oldanak meg a segítségével – e tevékenységük megfigyelése, illetve a résztvevők gondolkodási

folyamatának vizsgálata[6] szolgáltatja a kvalitatív adatokat, amelyek alapján az implementációs hibák vagy hiányosságok felfedezése, valamint – szerencsés esetben – az azok javítására tett javaslatok megtétele történik. A felderítő tesztekhez hasonlóan az értékelő tesztek elvégzését is ajánlatos minél hamarabb megkezdeni, annak érdekében, hogy a fejlesztés lezárultáig minél több fordulót sikerüljön végrehajtani.[7] Kiegészítő jelleggel kvantitatív adatok is gyűjthetőek a tesztelés során. Ugyan a jellemzően kis mintaméret, illetve a hangos gondolkodás okozta pontatlanságok[8] miatt ezek általában nem bizonyító erejűek – például az előző verzióval való összehasonlítás szempontjából –, a kvalitatív visszajelzések megértésében nagy segítséget nyújthatnak.

A harmadik fázisban az *értékelő* tesztek helyét *validációs* tesztek veszik át, amelyek már nem különálló komponensek halmazaként, hanem integrált egészként tekintenek az alkalmazásra, célja az eredeti elvárásoknak való megfelelés ellenőrzése, illetve a fejlesztés során felfedezett komolyabb problémák javításainak verifikálása. Ez általában számszerű értékekkel való összehasonlítás útján történik – rögzíthető például, hány percen belül kell a felhasználónak végrehajtani a feladatokat, hogy azokat sikeresként lehessen elkönyvelni –, vagyis ez a teszttípus főleg kvantitatív adatokra koncentrál; így jellemzően kevesebb fordulóval, fordulónként több résztvevővel kerül végrehajtásra. Új termékek esetén az első fázisban is célszerű lehet validációs teszteket végrehajtani, de nem a készülő, hanem néhány már meglévő, hasonló funkcionalitású alkalmazás vizsgálatával – az így gyűjtött adatok szolgálhatnak később mérceként az elkészült fejlesztés értékelésekor.[9]

A fentiek mind a korai tervezés, mind az implementáció során kiegészíthetőek *összehasonlító* tesztek elvégzésével, melyek a felmerülő lehetséges alternatívák közti döntést könnyíthetik meg, akár kvantitatív eredmények összehasonlításával – pl. azon opció mellett döntünk, ami általában a feladat gyorsabb végrehajtását tette lehetővé –, akár informálisan, a résztvevő felhasználók véleményét explicite kikérve és kiértékelve.

---

[6] Ennek érdekében a felhasználót természetesen meg kell kérni gondolatai kimondására, különösen, ha valahol elakadt, vagy nem olyan felülettel vagy eseménnyel találkozott, amire számított volna.

[7] Lévén a kvalitatív tesztek hatékonysága leginkább a fordulók számán alapul.

[8] Különösen, ha a feladat végrehajtásához szükséges idő is mérésre kerül, elvégre ettől nagyon nehéz különválasztani azt az időt, amit a felhasználó a meglátásai kifejtésével töltött.

[9] Meglévő termékek továbbfejlesztésekor általában célszerűbb az előző verzió már elvégzett validációs tesztjeinek eredményeit felhasználni az összehasonlítás alapjául.

3 A színházi kelléknyilvántartó rendszer

3.1 Bevezetés: strukturált és félstrukturált adatok kezelése

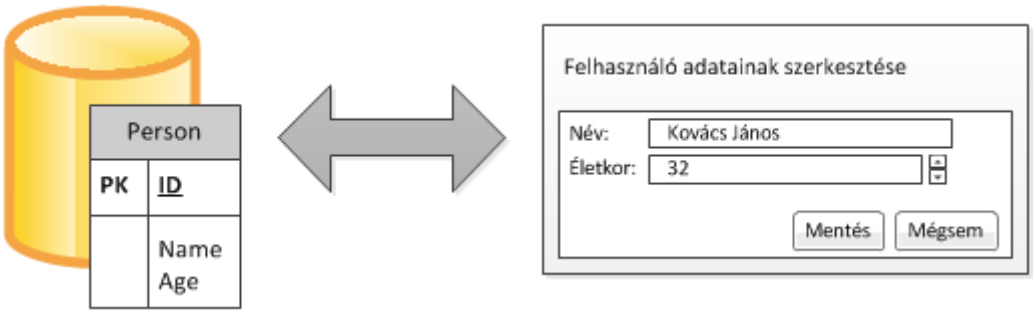
A vállalati információs rendszerek fejlesztése és üzemeltetése iránt mutatkozó egyre növekvő igénynek köszönhetően megnövekedett számú, adatvezérelt alkalmazások fejlesztésére irányuló szoftverprojektek a strukturált adatkezelést elősegítő tervezési minták, hardver-szoftver megoldások és a használhatóságot célzó bevett gyakorlatok kifejlődésének hatalmas lendületet adtak az utóbbi években. Napjainkban nem pusztán nagyvállalatok, hanem kis- és középvállalkozások is fizetőképes keresletet jelentenek mind az operatív, mind a stratégiai célú nyilvántartó alkalmazásokra, legyenek azok pusztán bekonfigurálást igénylő, kereskedelmi forgalomban készen kapható programcsomagok, vagy az egyedi fejlesztést elősegítő, általánosabb komponensek. Az adatok tárolását és karbantartását jellemzően relációsadatbázis-kezelő rendszerekre bízzák, amelyek gyakorlatilag minden elterjedt szerverplatformokon elérhetőek, és a grafikus felülettel rendelkező kliensrendszerekre is számos, kifejezetten az űrlapalapú felhasználást is nagymértékben elősegítő felhasználói interfész keretrendszer készült.

Az ilyen megoldások használatának közös, gyakran kimondatlan előfeltétele, hogy a kezelt adatok strukturáltak legyenek – vagyis a kezelendő adatok reprezentálhatóak legyenek entitások és kapcsolataik halmazaként, ahol mind az entitások, mint azok kapcsolatai korlátos számú, előre rögzített entitás- és kapcsolattípusok példányai, és a róluk tárolandó adatok hiánytalanul leírhatóak a típusukhoz definiált attribútum halmaznak megfelelő értékekkel.

Tárolás szempontjából a strukturáltság feltételének való megfelelés jelentősége abban áll, hogy lehetővé teszi, hogy a kezelendő adathalmaz leírható legyen a relációs adatmodell segítségével, így az erre felkészített, megbízható, nagy teljesítményű relációsadatbázis-kezelő rendszerek gondoskodhatnak ezek tárolásáról és karbantartásáról, hatékony módosításuk és lekérdezésük pedig a számos keretrendszer által támogatott SQL nyelv segítségével megvalósítható – szükségtelenné téve egyedi adathalmaz-bejáró logika kifejlesztését és optimalizálását.

A felhasználói felület kifejlesztése és felhasználóbaráttá tétele szintén nagymértékben egyszerűsödik statikus attribútum készlet definiálásával. Az adatok

rögzítése, módosítása és megjelenítése egyszerű űrlapokkal és táblázatokkal megvalósítható, melyek elkészítésére kiforrott keretrendszerek és programozói interfészek, valamint finomhangolásukra és felhasználóbaráttá tételükre jól bevált iránymutatások állnak rendelkezésre, mely utóbbiak betartásával potenciális felhasználók igen széles skálája számára biztosíthatunk már ismert, megtanult működést, viselkedést (4. ábra).



4. ábra: Strukturált adatok kezelésére mind az adat-, mind a prezentációs rétegben jól ismert, alaposan kidolgozott, széles körben elérhető megoldások állnak rendelkezésre.

Bizonyos esetekre azonban az említett megkötések nem alkalmazhatóak. Ha a tárolandó entitások nagymértékben különbözhetnek — az őket leíró attribútumok száma, típusa különböző —, a bemutatott megközelítés önmagában nem alkalmazható – sem szerveroldalon, a tárolást illetően, sem kliensoldalon, a felhasználói interfész szempontjából. Előbbire ugyan már kezdenek kialakulni és elterjedni platformfüggetlenül implementálható, jóldefiniált formátumok – például az eXtensible Markup Language (XML) leírónyelv, amelyet számos adatbáziskezelő szoftver támogat –, a kezelőfelületet illetően még nem beszélhetünk általános, hatékony, kényelmesen használható megoldásokról.

Ilyen eset például a színházi kellékek egységesített kezelése – vagyis jelmezek, díszletek, bútorok stb. tárolása ugyanazon adatbázisban, ugyanazon alkalmazáson keresztül –, ami nem valósítható meg előredefiniált entitások egy statikus halmazára szorítkozva. Minden kellék különböző típusú, számú, jelentésű attribútummal rendelkezhet, még ha hasonlónak tűnnek is – például két jelmez nem feltétlenül írható le relevánsan ugyanazon kérdések megválaszolásával, vagyis egy közös "jelmez" entitástípus definiálása, annak struktúrájára korlátozva az adattárolást, elkerülhetetlenül információvesztést fog okozni.

Bár bizonyos mértékű struktúráltság nélkülözhetetlen az adatok közös kezelésének és használatának biztosítására, félstrukturált tárolásuk már megadja az

ehhez szükséges formalizáltságot – lehetővé teszi például adatbázis-lekérdezések írását[10] –, mégis kellőképpen rugalmas a fentebb vázolt eset megoldására, mivel megengedi, hogy minden kellék teljesen egyedi attribútumhalmazzal rendelkezzen.[11]

3.2 Az adatrögzítő funkcióval szemben támasztott elvárások

A színházi kelléknyilvántartó rendszer célja egy általános, könnyen kezelhető elektronikus leltár biztosítása, nem csupán intézményen belül, de intézmények között is. A csatlakozó színházak egymás készleteiben kereshetnek, akár kölcsönözhetnek is egymástól. Hatalmas kelléktárat kell tehát átláthatóvá, rugalmasan kereshetővé tenni, illetve hatékony módot kell találni e nagy mennyiségű adat rögzítésére – alapvetően nem építhetünk már meglévő rendszerekre, azokból való importálás lehetőségére: hasonló leltárak az érintett intézményekben jellemzően nem, vagy csak szigorú szerkezet nélkül, informálisan (például papíron) készültek, melyeket várhatóan az új adatbázis kialakításakor nem lesz lehetőség érdemben felhasználni.

Mind az attribútumokat érintő rugalmasság, mind a hatékony kereshetőség követelményét szem előtt tartva, a kellékek tárolására a következő formát definiáltuk:

- A metaadatokat statikus struktúra szerint, hagyományos relációsadatbázis-oszlopok formájában adtuk meg. Ilyen metaadat például a kellék rövid megnevezése, kategóriája, fizikai azonosítója (például vonalkód vagy RFID tag) stb.
- Ezen felül minden kellék egyedi attribútumkészlettel rendelkezhet, amit XML formátumban tárolunk. Az attribútumok száma tetszőleges – a felhasználó minden fontosnak tartott információt rögzíthet egy-egy kellékről, név-érték párok formájában. (Egy attribútum neve lehet például „szín”, értéke pedig „piros”.)
- A kellékekhez képek is csatolhatóak, rövid leírással. Az egyes képeket különálló adatbázisrekordok reprezentálják, melyek külső kulccsal kapcsolódnak a kellékhez.

[10] Ezek a lekérdezések természetesen az SQL nyelv lecserélését vagy kiterjesztését igénylik, valamilyen, kifejezetten félstrukturált adatokhoz tervezett szintaxis használatával – mint amilyen például az XPath

[11] Beleértve az attribútumok számát és típusát is.

A rugalmas, hatékony kereshetőség igényét szem előtt tartva az attribútumok értékei típusosak, lehetővé téve például intervallum lekérdezések kiértékelését számadatokon, vagy szabadszöveges keresések végrehajtását szöveges értékeken. A színházi kelléknyilvántartó rendszerben négy alaptípust definiáltam:

- A *szöveges adat* típus kvalitatív értékek tárolására szolgál, szabadszöveges formában – így a legrugalmasabb, de a lekérdezések szempontjából legtöbb nehézséget okozó típus (figyelembe véve a lehetséges inkonzisztenciákat, amelyek az egyéni, informális megfogalmazásból eredhetnek, például eltérő szinonimák használata, elgépelések stb.).
- A *számadat* típus mérhető értékek tárolására használható. Formátuma a következő részinformációkból áll:
    - A mérés eredménye (SI egységben kifejezve), dupla pontosságú lebegőpontos szám formájában.
    - A mért fizikai mennyiség típusa, például „tömeg”, „hosszúság” vagy „terület”.[12]
    - A megjelenítés preferált mértékegysége. Míg az SI értékek egységes használata tároláskor segít az inkonzisztenciák, félreértelmezések elkerülésében, és lehetőséget biztosít összetett lekérdezések írására, fontos, hogy az értékeket prezentáláskor a felhasználó igényeihez igazítsuk, és számára kényelmes formát használjunk. Például, míg „0,45 m” és „45 cm” alatt ugyanazt a hosszúságot értjük, utóbbi természetesebbnek, emberibbnek hat, könnyebben érthető és értelmezhető, mint az SI mértékegység szerinti reprezentáció.
- A *dátum* típus időpontok tárolására szolgál. (Az időtartamokat számadatként rögzítjük.)
- A *komplex* típusú adat a négy alaptípusnak megfelelő értékek gyűjteménye (komplex típusú értékeket is beleértve). Ezt a típust a jövőbeni bővíthetőség lehetőségét szem előtt tartva vezettem be, jelenleg nem használjuk.

Míg a rögzített struktúrájú metaadatok megadása egyszerűen megvalósítható hagyományos űrlap vezérlőkkel – pl. statikusan elhelyezett szövegbeviteli mezőkkel –, használhatósági szempontból különösen nagy kihívás az egyedi attribútumok megadására szolgáló felület kialakítása. Rugalmasnak, gyorsnak és kényelmesen

[12] A lehetséges típusok halmaza előre definiált, de bővíthető.

kezelhetőnek kell lennie, hiszen a lényegi információ ezen attribútumok formájában kerül tárolásra az egyes kellékekről, így a felhasználói interfész e komponensének használata fogja kitenni a rögzítési folyamat legnagyobb hányadát; a kezdeti adatbázis kiépítésének időigénye, sőt, a rögzített adatok részletessége is múlik hatékonyságán.[13] Szintén lényeges elvárás a felhasználói hibák megakadályozása, észlelése és – alkalmas helyzetben – javítása, különösen a bemutatott, az átlagfelhasználó számítógépes tudásához és tapasztalataihoz képest meglehetősen összetett típusrendszerre való tekintettel: bármely típusú értéknél biztosítani kell a bevitel validációját, sőt, a megfelelő típus kiválasztásában is segítséget kell nyújtani. Utóbbi híján a felhasználók egy jelentős része várhatóan a kelleténél többször választaná a szöveges adat típust, köszönhetően egyszerűségének – különösen a meglehetősen összetett számadat típushoz képest. Nem megoldás ugyanakkor a típusrendszer teljes elrejtése sem. Bár mindenképp célravezető volna, ha az alkalmazás képes lenne az egyszerű felhasználói bemenetet értelmezni, és automatikusan meghatározni annak típusát, meg kell hagyni a lehetőséget ennek felülbírálására. Ezzel nem pusztán az alkalmazás helytelen döntéseinek helyrehozatalára van mód, de a felhasználói hibák detektálásában is nagy segítség lehet – a választott típus megjelenítése a felületen azonnali visszacsatolást jelent a beírt adat értelmezésére vonatkozóan, felhívva a figyelmet mind a formátumhibákra, mind az egyszerű elírásokra.[14]

3.3 A megvalósítás architekturális áttekintése

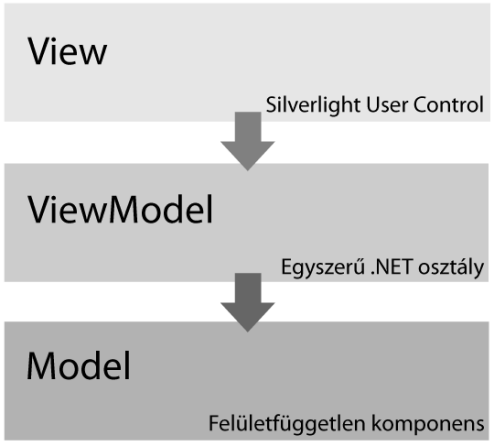
Szem előtt tartva mind a gazdag, reszponzív felhasználói élmény nyújtásának igényét, mind a könnyű telepíthetőség és a távoli elérés lehetőségének biztosítását, kliensplatform gyanánt a Silverlight mellett döntöttem. A Silverlight úgynevezett okoskliens platform: a vékonykliens megoldásokhoz hasonlóan alkalmazásonkénti telepítést nem igényel, a futtatáshoz csak böngészőre és egy azt kiterjesztő beépülő modulra van szükség. Ez utóbbi biztosítja az összetett futtatókörnyezetet, ami a böngésző beépített weblaprenderelő motorján és JavaScript virtuális gépén túl jóval

[13] Utóbbi oka, hogy egy kényelmetlenül használható attribútumbeviteli felület várhatóan a rögzített attribútumok számának csökkentésére ösztönözné a felhasználókat.

[14] A számadat típusú értékeket például meglehetősen kényelmetlen lenne teljesen manuálisan kitölteni, probléma esetén azonban – ha például az alkalmazás nem ismeri a felhasználó által megadott mértékegységet – a típus megjelenítése útján a hiba könnyen észrevehető és javítható.

összetettebb grafikus képességeket, nagytudású, típusbiztos osztálykönyvtárat és jobb teljesítményű felhasználói felületet tesz elérhetővé – mindez korábban csak vastagkliens alkalmazások készítésével volt lehetséges.

További előnye a Silverlight platformnak az igen fejlett adatkötés támogatás, ami a statikus űrlapok készítésének egyszerűsítésén túl az alkalmazás belső felépítésének lazán csatolt kialakításában is igen nagy segítséget jelent, lévén lehetővé teszi a Model-View-ViewModel (MVVM) tervezési minta használatát, ami a Model-View-Controller minta egy specializált változatának tekinthető. Ennek lényege, hogy a mélyebb alkalmazásrétegek, felületfüggetlen adatosztályok (Model) és azok nézete (View) közti kommunikáció, szinkronizáció közvetetté, ezáltal rugalmasabbá tételére ViewModel (VM) osztályokat vezetünk be (5. ábra). Ezek gyakorlatilag egy-egy nézet absztrakt reprezentációi: annak állapotát, viselkedését, működését írják le, és definiálják annak műveleteit, a konkrét vizualitás definiálása nélkül – így téve lehetővé a megjelenítés rugalmas változtatását, cseréjét (vagy akár teljes elhagyását, például unit tesztelés céljából), valamint a kód újrafelhasználását több azonos funkciójú, de eltérő kinézetű View között. Silverlight alkalmazások esetén implementálva ezt a mintát a ViewModel és a View szinkronban tartása jellemzően adatkötéssel és a Command minta használatával történik (ez utóbbi a Silverlight újabb verzióiban szintén erős adatkötés támogatást kapott), ami csökkenti a hibalehetőségek számát, ráadásul a VM osztálynak közvetlenül sosem kell hivatkoznia a konkrét felületelemekre, tovább növelve a megoldás rugalmasságát [17]. Mindez különösen fontos a felhasználói tesztekkel integrált fejlesztési projektek esetén: nagyon sok használhatósági probléma ugyanis a megjelenítés hiányosságaira vezethető vissza, így annak egyszerű finomhangolhatósága – az alkalmazáslogika módosítása nélkül – kulcsfontosságú.



5. ábra: A Model-View-ViewModel minta.

A kliensalkalmazás egy Windows Communication Foundation (WCF) alapú interfészen keresztül, szükség esetén, natív .NET komponensekkel is kiegészíthető – a színházi kelléknyilvántartó rendszer esetében erre az RFID- és vonalkódleolvasó eszközök meghajtóprogramjaival való kommunikáció biztosítása végett volt szükség.[15]

Szintén WCF használatával kapcsolódik a kliens az intézményi szerverhez, melyen ASP.NET 4.0 alapú webalkalmazás fut, ami a felhasználókezelést, adatelérési réteget és kapcsolódó üzleti logikát nyújtja. Az adatok perzisztenciája Entity Framework 4 segítségével történik SQL Server 2008 adatbázisba – utóbbi mellett erős félstrukturáltadat-támogatása mellett döntöttünk: a hagyományos, relációs adatok kezelése mellett képes XML adatok hatékony tárolására, indexelésére és lekérdezésére, amely képességeknek az egyedi attribútumok tárolása kapcsán vettük nagy hasznát.

Az egyes színházak közötti integráció egy Windows Azure alkalmazás segítségével történik. Ennek előnye, hogy – felhő alapú szolgáltatás lévén – az üzemeltetés és skálázás terhét nagyrészt leveszi az egyes intézmények válláról, valamint az elérhetőség, megbízhatóság is jóval nagyobb, mint egy saját fizikai szervergép telepítése esetén, ami e komponens központi jellege miatt különösen kritikus.[16]

Az alkalmazás architekturális felépítését szemlélteti a 6. ábra.

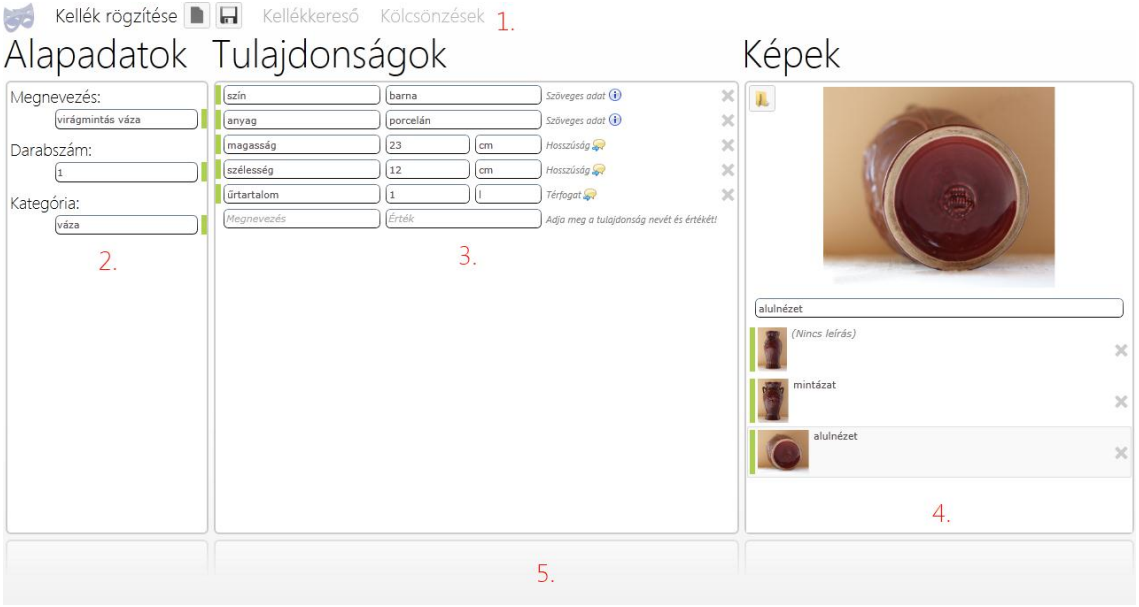
6. ábra: A színházi kelléknyilvántartó rendszer főbb komponensei

[15] Az alkalmazás természetesen ezen natív komponensek telepítése nélkül is fut, opcionális kiterjesztésekről van szó, az egyszerű böngészőből való futtatás lehetőségének megtartása mellett.

[16] Mindemellett a helyi színház szerverek képesek a globális komponens nélkül is működni – természetesen ekkor a többi intézménnyel való szinkronizálás lehetősége nem elérhető.

3.4 A rögzítő felület vizuális felépítése

A Silverlight alapú kliensalkalmazás rögzítő felületét a 7. ábra szemlélteti.



7. ábra: A színházi kelléknyilvántartó rendszer kellékrögzítő felülete.

A felhasználói interfész az alábbi főbb részekre bontható:

1. A felső menüsáv egyesíti a színházi kelléknyilvántartó rendszer kliensének összes funkcióját, valamint az aktív funkcióban elérhető parancsokat (az ábrán ez a rögzítés – a kereső és a kölcsönző jelenleg fejlesztés alatt áll).
2. Az *Alapadatok* panel szolgál a közös metaadatok megadására, hagyományos, statikus űrlap formájában.
3. A *Tulajdonságok* panelen rögzíthetőek a kellék egyedi attribútumai. (Az itt alkalmazott egyedi megoldásokra való tekintettel ez a funkció a továbbiakban részletesebben is bemutatásra kerül.)
4. A *Képek* panel a kellékről készített fényképeket gyűjti össze, valamint ugyanitt leírás is fűzhető hozzájuk.
5. Az alsó sáv a menüsor vizuális ellensúlyozására, a lényegi tartalom középre helyezésére szolgál – pusztán dekorációs célú.

Mind az *Alapadatok*, mind a *Tulajdonságok* panel beviteli mezőihez egyedi, háromállapotú ellenőrzést implementáltam. A hagyományos validációs megoldások kétállapotúak: a bemenet helyes vagy helytelen lehet, nincs különbségtétel a helytelenül megadott és a hiányzó adat között. A probléma ezzel a megközelítéssel a frissen megnyitott, még üres űrlapoknál jelentkezik: ilyenkor, ha a validáció azonnal lefut,

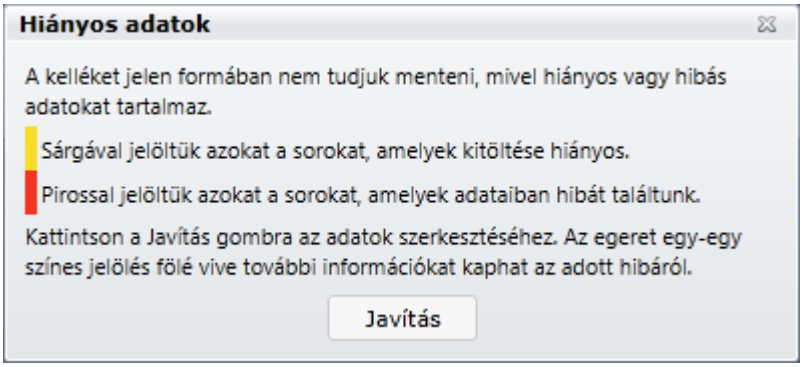
minden kötelezően kitöltendő mező hibát fog jelezni, azt az érzést keltve a felhasználóban, hogy még el sem kezdett dolgozni, de máris hibázott. A legtöbb felület ráadásul igen erős kiemelő színeket alkalmaz az ellenőrzési hibák jelölésére, amelyek – mivel ebben a helyzetben tömegével jelennek meg a képernyőn – így elvesztik figyelemfelkeltő hatásukat. A másik, gyakrabban alkalmazott megközelítés, hogy a validáció csak mentési kísérlet hatására fut le a teljes űrlapra, kitöltés közben pedig egyesével – a fókuszt éppen elveszítő mezőre – lép működésbe. Bár ez kiküszöböli az üres űrlapok imént ismertetett problémáját, inkonzisztens viselkedést mutat a felhasználó felé, hiszen az egyes mezők látszólagos helyessége más, a begépelt adatokkal közvetlen összefüggésben nem álló interakcióktól függ – nem ritka például, hogy az űrlap utolsó mezőjét tévesen hiszik helyesnek a felhasználók mentéskor, hiszen az nem veszíti el a fókuszt – és így nem validálódik – a *mentés* gombra kattintás előtt. Ezen kívül külön kell jelölni az ilyen jellegű megoldásoknál a kötelezően kitöltendő mezőket – erre vannak ugyan elterjedt megoldások (például a beírandó adat címkéjének megcsillagozása), ezek nem teljesen egységesek.

A háromállapotú validáció a vázolt problémát a *hiányzó* és *helytelen* adatok megkülönböztetésével küszöböli ki:

- A szövegmező mellett látható halványzöld jelölés fejezi ki a *helyesen* megadott információt – nem kötelezően kitöltendő adatok esetén már az alapértelmezett, üres állapotban is ez a szín látható.
- Sárga jelölés mutatja a *hiányzó* adatot; kötelezően kitöltendő mezők mellett jelenik meg – már az űrlap megnyitásakor is – amennyiben azok üresek. Így minden időpillanatban helyes tájékoztatást kaphat a felhasználó az általa megadott értékek helyességéről (vagy éppen hiányáról), anélkül, hogy a hibázás érzetét keltenénk benne.
- Élénk, piros jelölés mutatja a *hibásan* kitöltött mezőt, például ha számot rögzítő mezőbe nem várt karakter kerül, vagy más formátumhibát észlelünk.

Mivel a felület kevés színt alkalmaz, az említett jelzések – különösen a legutolsó – kis méretük ellenére kiemelkednek, így a hibák, hiányosságok könnyen észrevehetőek. Sőt, a formátum szempontjából helyesen megadott információk áttekintésében is segíthetnek mentés előtt, mivel a zöld jelölések elkülönítik a felhasználói bemenetet a többi grafikus elemtől.

Egy-egy hiba, hiányosság oka az egérkurzort a jelölés fölé mozdítva, az eszköztippben tekinthető meg részletesen – erre az alkalmazás fel is hívja a figyelmet mentéskor, amennyiben egy vagy több jelölő állapota eltér a *helyestől* (8. ábra).



8. ábra: Validációs problémák részletezése mentéskor.

3.5 Egyedi attribútumok rögzítése

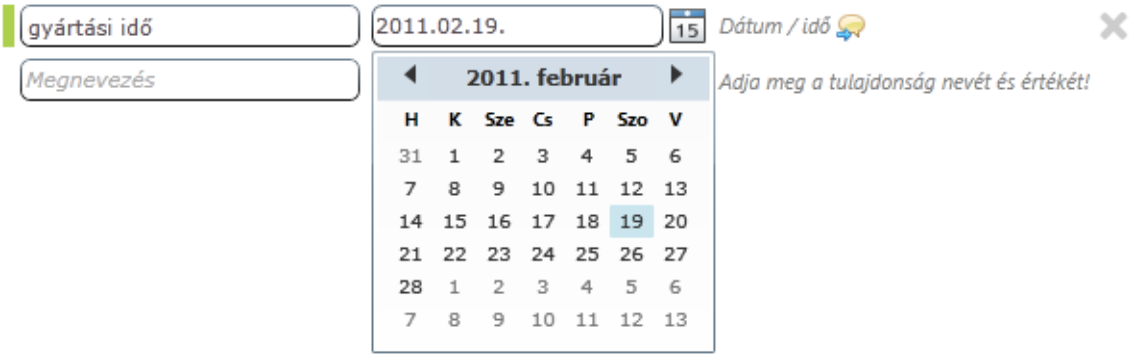
Az egyedi attribútumok rögzítésére a 9. ábrán látható felületet alakítottam ki.

9. ábra: Egyedi attribútumok rögzítése.

Ahogy az ábra 3. sorában látható, minden attribútum megadása – típustól függetlenül – két szövegdoboz kitöltésével történik: az egyik a tulajdonság nevének, a másik az értékének kitöltésére szolgál. Az adatok beírása után a jobb szélen látható gomb megnyomásával, vagy az Enter billentyű leütésével megtörténik az attribútum rögzítése. Ekkor az alkalmazás – reguláris kifejezések segítségével – meghatározza a tulajdonság típusát, majd automatikusan két új, üres szövegdobozt illeszt a felületre a következő attribútum beírásához (a példában ezek a negyedik sorban jelennének meg) – így az adatok bevitele ugyanolyan gyorsan megtörténhet, mint hagyományos, statikus űrlapok esetén: a felhasználónak nem kell az egérhez nyúlnia, vagy más, munkáját hátráltató extra lépéshez folyamodnia, hogy bővítse az attribútumok halmazát.

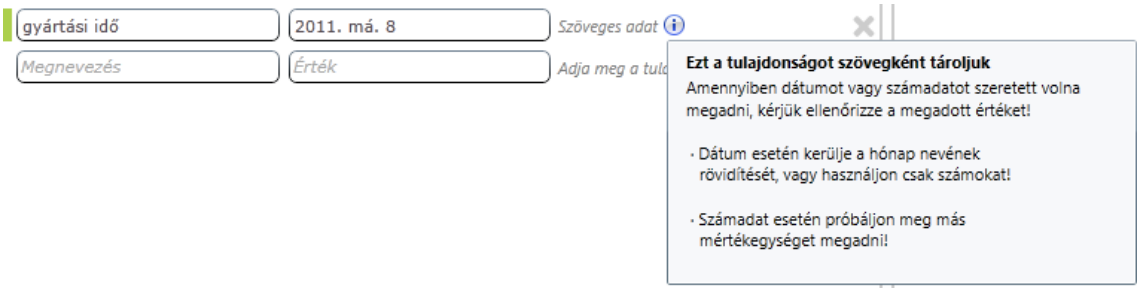
A már rögzített tulajdonságok típusa jól látható helyen – a sor végén, ahova a felhasználó tekintete fókuszál az Enter leütésekor – jelenik meg, így könnyen észrevehető, ha bármelyik is eltér a felhasználó eredeti szándékától. A típus ezen felül az attribútumra vonatkozó felületelemek elrendezését is befolyásolja: míg szöveges adat esetén módosításra is ugyanúgy két szövegdoboz szolgál, mint bevitelkor,

számadatoknál a mértékegység külön mezőbe kerül (9. ábra 2. sora) – az áttekinthetőség elősegítése és az esetleges validációs problémák javításának egyszerűsítése végett –, dátumok szerkesztésére pedig egy beágyazott naptár vezérlő is használható (10. ábra).



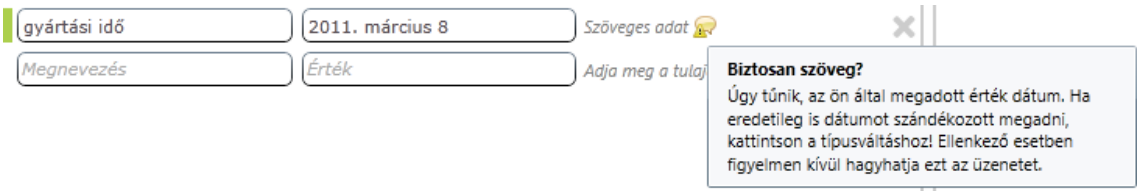
10. ábra: Dátum típusú attribútumok szerkesztése.

Amennyiben az alkalmazás a típust a felhasználó szándékától eltérően határozta meg, lehetőség van annak módosítására. Minden, nem szöveges típusú attribútum mellett megtalálható a *típuskonverzió* ikon (jelenlegi megjelenése egy szövegbuborékra mutató nyíl), amire kattintva a bemenet visszakonvertálható szöveges adattá.[17] Ellenkező esetben, ha a felhasználó *számadat* vagy *dátum* megadását kezdeményezte, és azt mégis szövegként értelmezte az alkalmazás, a sor végén megjelenő információs ikon eszköztippje nyújt segítséget a helyes formátum megválasztásában (11. ábra). A bemenet javítását követően a felület felajánlja az immár helyesen felismert típusra való konverziót, így ez a probléma is gyorsan orvosolható. (12. ábra)



11. ábra: Segítség a formátumhibák javításához.

[17] Mindez természetesen vissza is vonható.



12. ábra: A formátum javítása után a helyes típusra történő konverzió egy kattintással elvégezhető.

3.6 Az alkalmazott tesztelési folyamat

A rögzítő felület első változatának elkészültével *értékelő* tesztet hajtottam végre, részben azt ellenőrizendő, hogy a korábban bemutatott, félstrukturált tárolási koncepció – legalábbis amennyit ebből a felhasználó felé kommunikálunk, és amennyire szükséges is ismernie az alapokat a kellékek beviteléhez – valóban érthető, megfogható-e akár az alacsony számítógépes ismeretekkel rendelkező felhasználók számára is, valamint alkalmas-e a releváns információk részletes és hatékony bevitelére. Ezenkívül arra is kíváncsi voltam, a konkrét megvalósítás felhasználói élménye milyennek bizonyul, lehetővé teszi-e a kényelmes és gyors munkát.

Kiegészítő jelleggel kvantitatív adatokat is gyűjtöttem. Erre való tekintettel a tesztet 10+1 felhasználóval hajtottam végre[18] – bár ezek a mérések a hagyományos, matematikai statisztikán alapuló kiértékelést nem teszik lehetővé, ahhoz már elegendőek, hogy összehasonlítási alapot szolgáljanak a jövőbeni továbbfejlesztett változatok megítéléséhez [18]. A számszerű adatok nagy részét az alkalmazás rögzítette eseményekből származtattam (például kellék beviteléhez szükséges idő, hozzáadott attribútumok száma, képek száma, ebből hányhoz készült leírás stb.), a teszt végén azonban egy rövid kérdőívet is kitöltettem a résztvevőkkel, hogy az általuk megélt felhasználói élményről is képet kapjak. Ez utóbbi méréshez a System Usability Scale (rendszer használhatósági skála) néven ismert kérdéssort használtam [19]. A kitöltőnek 10 állítással kapcsolatban kell kifejeznie egyetértését – vagy egyet nem értését – 1-től 5-ig terjedő skálán, ahol előbbi az „egyáltalán nem értek egyet”, utóbbi a „teljes mértékben egyetértek” megfelelője. Az értékelendő állítások a következőek:

1. Azt hiszem, szívesen használnám ezt az alkalmazást rendszeresen.
2. Az alkalmazást szükségtelenül bonyolultnak találtam.

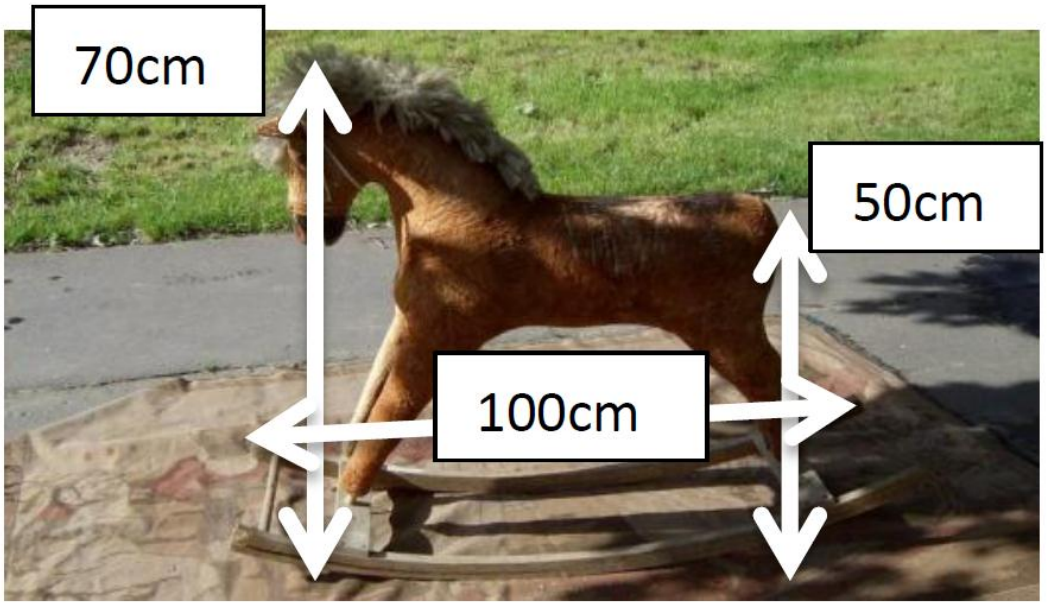
[18] Az első felhasználóval inkább a teszteléshez kitűzött feladatokat, mint magát az alkalmazást teszteltem – a feladatok azonban már első alkalommal érhetőnek és relevánsnak bizonyultak (nem kellett módosítanom őket), így az akkor gyűjtött adatokat is felhasználtam.

3. Az alkalmazást könnyen használhatónak éreztem.
4. Úgy gondolom, az alkalmazás használatához technikai személyzet segítségére lenne szükségem.
5. Az alkalmazás funkcióit jól integráltnak találtam.
6. Úgy gondolom, az alkalmazásban túl sok a következetlenség.
7. Szerintem a legtöbb ember nagyon gyorsan megtanulná az alkalmazás használatát.
8. Nehézkesnek találtam az alkalmazás használatát.
9. Magabiztosnak éreztem magam az alkalmazás használatakor.
10. Sok mindent kellett megtanulnom az alkalmazás használatának megkezdéséhez.

Mint látható, az állítások fele pozitív, fele negatív megfogalmazású; ezek váltakozása hivatott elősegíteni a monotónia elkerülését – mindez várhatóan a válaszok mélyebb átgondolására ösztönöz.[19] A kérdéssor kiértékelése az ún. SUS pontszám kiszámításával történik, ami 0-tól 100-ig terjedő skálán helyezi el a felhasználói élmény jóságát (ahol a nagyobb érték pozitívabb élményt jelent).

A kvalitatív adatok gyűjtése a már ismertetett módon, a felhasználók hangos gondolkodásra kérésével történt, hang- és képernyőrögzítő szoftverek segítségével. A megoldandó feladat néhány színházi kellék bevitele volt. Kifejezetten ügyeltem arra, hogy a feladatlapon kerüljem a dolgok megnevezését: sem a kellékeknek, sem azok mért tulajdonságainak nem adtam nevet, elvégre a kész alkalmazás leendő felhasználóinak erre is képesnek kell lenniük, méghozzá magukkal és egymással konzisztens módon. A rögzítendő kellékekre egy példa a 13. ábrán látható – a feladatlap ilyen és ehhez hasonló ábrák egymásutánjából állt, minden további szöveges információ nélkül. Kaptak továbbá a felhasználók egy pendrive-ot is, amelyen további képeket találhattak a kellékekről, de ez is nélkülözött bármiféle struktúrát vagy megnevezést (minden kép egyetlen mappába volt elhelyezve, fájlneveik számok voltak).

[19] Ha például minden állítás pozitív megfogalmazású, a kitöltők hajlamosak néhány válaszadás után egy „átlagos” pontszámot kialakítani magukban, és minden további állításra azzal válaszolni.



13. ábra: A teszt során rögzítendő kellékek mérhető adatainak leírása során ügyeltünk arra, hogy a feladatlap ne befolyásolja a résztvevőket a tulajdonságok megnevezésében.

Az alkalmazás kipróbálása előtt a felhasználók rövid eligazításban részesültek a szituációt illetően (amelynek során az alkalmazás szerepe és célja röviden ismertetésre került), valamint egy rövid videót is megtekintettek a szoftver használatának alapjairól – nem célunk ugyanis, hogy a rögzítő felület minden előismeret nélkül, pusztán próbálgatás, felfedezés útján használható legyen, elvárás ugyanakkor, hogy alapvető elsajátítása ne is igényeljen többet egy néhány perces, egyszerű bemutatónál.[20] A kellékek rögzítését követően a teszt a már ismertetett SUS kérdőív kitöltésével zárult.

[20] A videofelvétel egy példa kellék rögzítését mutatja be röviden, minden egyéb háttér-információ (például diavetítés) nélkül – célom az volt, hogy a leendő felhasználók akár egymásnak is képesek legyenek megmutatni alapokat. A teszt során azért döntöttem mégis a videofelvétel mellett, hogy biztosíthassam, mindenki pontosan ugyanazzal az alaptudással kezd neki a feladatmegoldásnak.

4 Az eredmények értékelése és alkalmazása

4.1 Áttekintés

Összességében elmondható, hogy a felhasználók mindegyike hamar megértette a kellékrögzítő felület működését, és a mögötte álló félstrukturált koncepciót: képesek voltak egyedi attribútumok segítségével rendszerezni a kellékek releváns adatait, és hatékonyan rögzíteni azokat. A felhasználói interfész kényelmi szolgáltatásai is hasznosnak bizonyultak számukra – például mindegyikük kerülte az egér használatát a tulajdonságok bevitele során, hamar megszokták a billentyűzettel történő rögzítést, és a típusfelismerés is az elvárásoknak megfelelően működött: az elgépeléseket és ritka formátumhibákat leszámítva mindig a felhasználó szándékainak megfelelő jelentést következtette ki az alkalmazás. A kivételt jelentő néhány eset sem zökkentette ki a résztvevőket: a felület visszajelzéseinek köszönhetően hamar észrevették a problémát, önállóan korrigálták a formátumot, és a hiba javítását is hamar el tudták végezni a kézi típuskonverzió segítségével – csupán a résztvevők egyikének akadt nehézsége az egyik, általa használni kívánt, de az alkalmazás számára ismeretlen formátum lecserélésével, a hangos gondolkodásnak köszönhetően azonban ez esetben is könnyen a probléma végére tudtam járni.[21]

A kvantitatív adatok összesítése során három szempont szerint jelöltem ki összehasonlítási alapot az elkövetkező verziók számára: hatékonyság, részletesség és felhasználói élmény. Az első kettőre vonatkozó eredményeket tartalmazza az 1. táblázat: a teljes kellékkészlet rögzítéséhez szükséges időt, illetve az ennek során bevitt attribútumok számát szemlélteti, résztvevőkre lebontva. Utóbbi mérőszám szerepe kettős. Egyrészt kifejezi, mennyire motiváló újabb és újabb tulajdonságokkal bővíteni egy-egy kellék adatlapját – amennyiben jelentős csökkenést tapasztalnék ezen értékekben a későbbi verziók tesztelése során, mindenképpen meg kell majd vizsgálnom, nem vezettek-e be a módosítások olyan új kényelmetlenségeket vagy nehézségeket, amelyek kevésbé részletes adatbevitelre ösztönöznék a felhasználókat.

[21] A probléma oka valójában nem is a formátumban, hanem a mögöttes típusrendszerben rejlett: az érintett résztvevő intervallum adatot szeretett volna rögzíteni egyetlen attribútum segítségével, ami a rendszerben jelenleg ismeretlen fogalom. A következő szakaszban erre a kérdésre részletesen is kitérek.

Másrészt a végrehajtáshoz szükséges idők összehasonlítása sem történhet meg e számok ismerete nélkül – figyelembe véve, hogy több attribútumot rögzíteni értelemszerűen több időbe kerül. Erre tekintettel bevezettem egy származtatott mérőszámot, az attribútumonkénti időigényt, ami már számításba veszi az ebből eredő eltéréseket, és így jóval használhatóbb összehasonlítási alapként szolgál.

1. táblázat
Hatékonysági mérések eredményei

Résztvevő (P a próbateszt résztvevője)	Az összes kellék rögzítéséhez szükséges idő (perc)	Egyedi attribútumok száma	Attribútumonkénti időigény (másodperc)
P	30.4	71	26
2	46.9	59	48
3	24.2	41	35
4	46.1	70	40
5	28.0	63	27
6	28.4	54	32
7	30.7	49	38
8	28.6	63	27
9	63.1	113	33
10	32.9	52	38
11	20.4	43	29
Átlag	34.5	61.6	34

A felhasználói élmény mérésére, a már ismertetett módon, a System Usability Scale-t használtam – ennek eredményeit a 2. táblázat összesíti. A tesztelt felület átlagos SUS pontszáma kifejezetten magas, 89.3 lett a résztvevők körében.

2. táblázat
SUS kérdőívek eredményei

Résztvevő (P a próbateszt résztvevője)	Állítások										SUS pontszám
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	
P	5	1	5	1	5	1	5	2	5	1	97.5
2	5	1	5	1	5	1	5	2	4	1	95.0
3	2	1	4	1	2	4	5	1	2	1	67.5
4	4	1	5	1	5	1	5	1	5	1	97.5
5	3	1	4	1	5	1	4	1	3	1	85.0
6	4	1	5	1	5	1	5	1	4	1	95.0
7	4	1	4	1	4	2	4	2	3	1	80.0
8	4	1	4	2	5	2	4	1	4	3	80.0
9	4	1	5	1	5	1	5	1	5	1	97.5
10	5	1	5	1	5	1	5	4	4	1	90.0
11	5	1	5	1	4	1	5	1	5	1	97.5

4.2 A tesztek hatására implementált módosítások

Az egyik leggyakoribb kritika, ami szinte minden résztvevőnél elhangzott, a kategóriarendszert illette: a tesztelt változat mindössze egyetlen szövegbeviteli mezővel szolgált egy-egy kellék kategorizálására – a felhasználók szükségét érezték volna valamilyen támpontnak arra vonatkozólag, mennyire legyenek specifikusak a csoportosítást illetően (például a fakardot kardként vagy fegyverként osztályozzák-e), illetve milyen neveket adjanak kategóriáiknak (például kard vagy kardok). Bár végül mindannyiuknak sikerült döntésre jutnia e kérdéseket illetően, ahogy a 3. táblázat is mutatja, a probléma valós, és nem kevés inkonzisztenciát okozott.

3. táblázat
A kellékek rögzítéséhez használt kategóriák, illetve az egyes kategóriákat használó résztvevők száma

Kategória	Résztvevők száma (fő)	Kategória	Résztvevők száma (fő)	Kategória	Résztvevők száma (fő)
asztalteríték	1	játék	2	konyhai kellékek	1
bútor	1	játékok	1	lakberendezés	1
díszítés	1	jelmez	8	lámpa	6
edény	1	jelmezek	2	női kalap	1
étkészlet	3	kalap	4	női ruházat	1
evőeszköz	5	kalapok	1	öltözet	1
fegyver	4	kanál	1	ruha	1
fejfedő	2	kancsó	2	ruha kiegészítő	1
gyerekjáték	3	kanna	2	tányér	4
gyerekjátékok	1	kard	4	télapójelmez	1
harci eszközök	2	kardok	1	világítás	2
hintaló	4	konyha	1	villa	1
hintalovak	1	konyhai eszközök	3		

Az egységesítést elősegítendő, az egyszerű szövegbeviteli mező a következő verzióban automatikus kiegészítés funkcióval bővült, ami a korábban már használt kategórianeveket ajánlja fel új kellékek rögzítésekor. A módosítás, ha a specifikusság kérdésében nem is, a név inkonzisztenciák elkerülésére várhatóan megoldást fog jelenteni. A probléma teljes megszűntetése valószínűleg adatmodell szintű változtatást is igényel majd – erről a következő szakaszban bővebben is lesz szó.

További nehézséget jelentett az azonos kategóriába tartozó kellékek tulajdonságainak konzisztens megválasztása: bár a felhasználók törekedtek arra, hogy a hasonló tárgyakat hasonló nevű attribútumokkal írják le, a felület tesztelt változata nem nyújtott ebben segítséget. Ezért az új verzióban az attribútum nevének szövegdoboza is kapott automatikus kiegészítés funkciót: a felület kilistázza azon tulajdonságok megnevezését, amelyeket a felhasználó az aktív kellékhez még nem rögzített, de a korábban felvitt, az aktuálissal megegyező kategóriájú[22] kellékek tartalmaznak (14. ábra). Szöveges attribútumok esetén az érték mező is rendelkezik ezzel a funkcionalitással, vagyis a korábban használt értékek felajánlásával (15. ábra).[23]

[22] Kihasználva, hogy a kategória megadása a tulajdonságok bevitele előtt történik, így a felület számára ismert információ.

[23] Figyelembe véve a tulajdonság nevét és a kellék kategóriáját.

14. ábra: Automatikus kiegészítés a kellék tulajdonságainak megnevezésekor.

15. ábra: Automatikus kiegészítés szöveges attribútum értékének megadásakor.

A tesztek rávilágítottak olyan kisebb, fejlesztők számára nehezen észrevehető, de felhasználókat nagyon zavaró hiányosságokra is, mint például rosszul beállított TAB sorrend, megerősítő dialógusok hiánya bizonyos helyeken (például kép törlésekor), vagy annak egyértelműsítése, mely menüpontok tartoznak egy csoportba (a módosított menüt szemlélteti a 16. ábra).

16. ábra: A menüpontok csoportosítása vizuálisan is megjelenik.

Ezen kívül, az iteratív modellnek megfelelően, a felületet olyan funkciókkal is bővítettem, amelyek bár a tesztelés során nem érzékeltették hiányukat, az eredeti követelményrendszernek részei – ilyen a fizikai azonosítók kezelése, valamint a webkamera támogatás. Ezek egy valós kelléktárban mindenképpen hasznos és fontos lehetőségek, az első teszt során azonban nem lett volna szerepük, lévén előbbihez kész vonalkód- vagy RFID olvasóra, utóbbihoz pedig kézzel fogható teszt kellékekre lett volna szükség. A második tesztelési ciklusban azonban – a fent bemutatott javításokkal, módosításokkal együtt – ezek kiértékelése is meg kell, hogy történjen.

4.3 Koncepcionális továbbfejlesztési javaslatok

Az eddig bemutatott fejlesztéseken és javításokon túl a kvalitatív visszajelzések elemzése során mélyebb, a felhasználói felületen kívül tárolási struktúrát is nagymértékben befolyásoló módosítási lehetőségek is felmerültek. Ezek hosszabb távon, komolyabb átgondolást és áttervezést követően kerülhetnek bele az alkalmazásba – sem szükségességük, sem megtérülésük nem triviális, különös tekintettel arra, hogy – nagyobb léptékű változások lévén – magukban hordozzák a felhasználói élmény

romlásának veszélyét, például az egyszerűséget, hatékonyságot vagy áttekinthetőséget illetően.

Az első ilyen koncepcionális kérdés a kategóriarendszer már ismertetett problémája. A jelenlegi megoldás tervezésének fő szempontjai az egyszerűség és adaptivitás voltak – vagyis, hogy a felhasználók ne töltsenek túl sok időt kategorizálással, valamint hogy ne előre rögzített kategóriahalmazból kelljen választani, az opciók illeszkedjenek a valós igényekhez, az egyes intézmények mindenkori kellékkészletéhez. Három felhasználó is felvetette azonban, hogy szívesebben dolgozna hierarchikus kategóriákkal – amelyek valóban megoldanák a specifikusság korábban bemutatott problémáját, de jóval összetettebb felületet igényelnének –, illetve öten érveltek az előre rögzített kategóriák egyszerűsége és kényelmessége mellett[24] – át kell gondolni azonban, valóban lehetséges-e ezeket előre, és nem a kellékek leltározása során összegyűjteni, és hogy egy ilyen statikus lista mennyire bizonyulna kifizetődőnek hosszabb távon.

Szintén módosítást igényelhet a darabszámok kezelése. Jelenleg minden kellék metaadatában szerepel, mennyi áll rendelkezésre belőle – például egy doboz pingponglabdát nem külön-külön rögzítünk, azonosítunk stb., hanem egyetlen kellékként szerepeltetjük az adatbázisban, metaadatban jelezve, hány darabos készletről is van szó.[25] A felhasználóknak azonban nem minden esetben volt egyértelmű e mező helyes használata. Ugyanis, kifejezetten ezt ellenőrizendő, a tesztben szerepelt a 17. ábrán látható tányérkészlet – mely háromféle tányért tartalmaz, az egyes fajtákból ráadásul nem azonos számút –, melynek bevitelét illetően több, igen különböző megközelítést figyeltem meg:

Egyetlen kellékként 8 felhasználó rögzítette a készletet („tányérkészlet”, „étkészlet” vagy hasonló néven). Ebben az esetben a darabszám mező értéke – a mi definíciónk szerint – 1 kellene legyen, lévén a rögzített kellék a készlet, és nem a tányér (mely utóbbiból több, előbbiből azonban csak egy van). A 8 felhasználóból 3 gondolkodott hasonlóképpen, 5-en viszont a darabszám mezőbe a készletben található tányérok számát írták (annak ellenére, hogy azok nem ugyanolyanok, illetve

[24] Közülük ketten explicite hozzátették, hogy a lista azért legyen bővíthető.

[25] Elsősorban a fizikai azonosítás miatt döntöttünk e megközelítés mellett: míg a dobozra könnyen ragasztható vonalkód, minden egyes pingponglabdát külön-külön ellátni ilyen azonosítóval összehasonlíthatatlanul nagyobb munka, ráadásul a leltárt is áttekinthetetlenebbé teszi.

megnevezés is készletről, nem tányérról szólt). Továbbá a 8 felhasználóból 4 fejtette ki pontosabban is, melyik tányérfajtából hány darab található a készletben – jobb híján *számadat* típusú attribútumok formájában (pl. „mélytányér darabszáma"). E 8 felhasználónak ráadásul meg kellett oldania a fajtánként eltérő tulajdonságok helyes rögzítését – ezt jellemzően az attribútum nevének prefixálásával oldották meg (pl. „kistányérok átmérője", „lapostányérok átmérője" stb.).

Külön kellékként 3 felhasználó rögzítette az egyes tányérfajtákat – felismerve, hogy ahhoz, hogy 1-től eltérő darabszámot írhassanak, minden elem teljesen egyforma kell legyen. E megoldás hátránya, hogy azon tulajdonságokat többször is be kell írni, amelyek az egyes tányérfajtákra ugyanolyanok (pl. szín, anyag), illetve – ahogy az egyik résztvevő meg is jegyezte – ezáltal elvész az az információ, hogy ezek a tányérok tulajdonképpen egy készletbe tartoznak.

17. ábra: A darabszám mező használatának vizsgálatát célzó kellék, ahogy a feladatlapon szerepelt.

Végig kell gondolni, hogy a későbbi felhasználás – például keresés, kölcsönzés – szempontjából melyik megközelítés bizonyulna a legcélszerűbbnek, szükség esetén hozzá kell igazítanunk az adatmodellt, és a felületen egyértelműbbé kell tennem a vonatkozó mezők helyes kitöltésének módját.

Az egyedi tulajdonságok rögzítését illetően az észlelt formátumhibák – bár a legtöbb esetben csak elgépelést jelentettek – bizonyultak hasznos indikátornak. Rávilágítottak, milyen típusú adatokat szerettek volna rögzíteni a felhasználók az

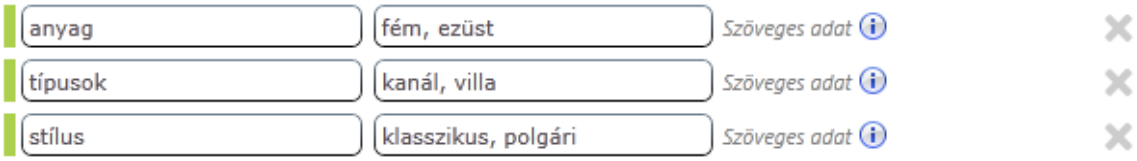
általam definiáltakon túl, mi több, azt is megfigyelhettem, milyen formátummal adnák meg ezen attribútumokat, ha az alkalmazás támogatná őket.

A következő adattípusok bevezetésének lehetőségét volna célszerű megvizsgálni:

- *Intervallumadat:* az intervallumok tárolási struktúrája két szám, melyek a minimális és a maximális értéket fejezik ki. Két résztvevő szeretett volna ilyen tulajdonságot megadni, egyikük meg is próbálta. Az általa használt formátum „<min>-<max>”, ahol <min> a minimális, <max> a maximális érték, és mindkettő formátuma megegyezik a *számadat* formátumával.
- *Hozzávetőleges adat:* ez a típus a meglévő *számadat* kibővítése egy bittel, ami azt jelzi, pontos-e az érték vagy nem. Négy felhasználó szeretett volna ilyen tulajdonságot rögzíteni, hárman közülük a következő formátumot javasolták vagy próbálták meg: „kb. <szám>”, ahol <szám> formátuma megegyezik a *számadat* formátumával.
- *Lista:* ez a típus többértékű attribútumok használatát tenné lehetővé: például ha egy kellék többszínű, többféle anyagból készült stb. Gyakorlatilag a tárolási rétegben már definiált, de a felületre nem kivezetett *komplex* típus egyszerűsített változata, amennyiben a *lista* típus nem engedi meg, hogy értékeinek típusa különbözzön (míg a *komplex* ezt lehetővé teszi). Hét felhasználó adott meg ilyen jellegű tulajdonságot, az értékeket vesszővel elválasztva sorolták fel (18. ábra).

Megjegyzendő, hogy míg az első és az utolsó javasolt típus gyakorlatilag a már definiált *komplex* típus speciális eseteinek tekinthetőek, így hiányuk látszólag elkerülhető lett volna, ha azt már az első változatban kivezettem volna a felületre, nem véletlen, hogy kihagyása mellett döntöttem. Ilyen típusú adatok megadásához és szerkesztéséhez ugyanis igen összetett felhasználó interfészre lett volna szükség – túl bonyolult lett volna ahhoz, hogy a többi típushoz hasonló hatékonysággal, egér és unintuitív billentyűkombinációk segítsége nélkül használható legyen. Ennek legfőbb oka éppen túlzott általánosságában rejlik: hogy nincs definitív, jól meghatározott szerepe a kellék tulajdonságainak leírásában. A felhasználói tesztek azonban megmutatták, milyen helyzetekben lehet rá szükség, és hogyan lehet struktúráját specializálni, e helyzetekhez igazítani, hogy használata hatékony, kényelmes és a többi típussal konzisztens legyen: vagyis két szövegmező, és egy jól meghatározott formátum

elégséges legyen rögzítéséhez. Mi több, a konkrét formátumok definiálását illetően is segítséget nyújtottak a visszajelzések.



18. ábra: Lista jellegű attribútumok, ahogy a 9. sz. résztvevő kísérelte meg rögzíteni – mivel ezt a formátumot a felület nem ismeri, tárolásuk szöveges adatként történt.

5 Összefoglalás

A dolgozatban áttekintettem a felhasználói felület fejlődésének mérföldköveit és jelentőségét napjaink kliensalkalmazásaiban, majd bemutattam a korszerű GUI fejlesztés főbb nehézségeit, problémáit, technikai és munkafolyamat szinten egyaránt; a továbbiakban pedig ismertettem a főbb az elvárásokat, amelyeknek napjaink interfészei meg kell, feleljenek – a terület nagy úttörői, a Xerox és az Apple eredményei alapján.

Mindezek után könnyen láthatóvá vált, miért is tulajdoníthatunk különös jelentőséget a felhasználói tesztelésnek – valós képet a használhatóságról, a felhasználói élményről ugyanis kizárólag oly módon szerezhetünk, ha megfigyeljük, hogyan teljesít az alkalmazás valósághű környezetben. Bár a visszajelzések gyűjtésének e formája korábban drága és időigényes volt, a Jakob Nielsen javasolta módosítások és egyszerűsítések bevezetésével – a kvantitatív mérésekről a kvalitatív visszacsatolásra helyezve a hangsúlyt – már néhány résztvevővel is értékes és jól használható eredményekkel szolgálhat, amennyiben a tesztelés kellően korán megkezdődik, és az életciklus során többször is lezajlik. Ennek lehetőségeit is bemutattam a 2. fejezetben.

A következő szakaszban egy esettanulmányon keresztül demonstráltam az ismertetett metódust. Míg a legtöbb adatvezérelt alkalmazás strukturált, relációs adatmodellre egyszerűen leképezhető entitásokkal dolgozik, bizonyos helyzetekben rugalmasabb tárolásra van szükség, ami a felülettervezést is újszerű kihívás elé állítja. Ilyen például a színházi környezet: a színházi kelléknyilvántartó rendszer, a legtöbb leltári alkalmazással ellentétben, félstrukturált adatokat kell kezeljen, amelynek felhasználó felületre való kivezetése túlmutat a hagyományos űrlapalapú alkalmazások bevált megközelítésén. Nem valósítható meg jól ismert, alapszintű vezérlők – mint amilyen a szövegdoboz, vagy a legördülő lista – egyszerű, statikus elrendezésével.

A dolgozatban áttekintettem a színházi kelléknyilvántartó rendszer architektúráját, majd bemutattam dinamikus, mégis egyszerűen kezelhető megoldásomat a félstrukturált adatok rögzítésére szolgáló felhasználói interfész megvalósítására, kitérve annak hatékonyságot növelő és tanulhatóságot elősegítő kényelmi szolgáltatásaira is. A koncepció helyességét, érthetőségét, valamint a felület használhatóságát felhasználói tesztek segítségével ellenőriztem és értékeltem ki.

A 4. fejezetben az elvégzett tesztek eredményeinek vizsgálatát taglaltam. A kapott visszajelzések tanúsága szerint a felhasználók mindegyike hamar megértette a kellékrögzítő felület működését, és a mögötte álló félstrukturált koncepciót, valamint a kényelmi szolgáltatásoknak is jó hasznát vették. A kvalitatív méréseim is ezt a benyomást erősítik: mind a hatékony munkavégzés, mind a pozitív felhasználói élmény elérésében sikeresnek bizonyult a grafikus interfész.

Megoldásom verifikálásán túl a fejlődés jövőbeni irányának kijelölésében is hasznos segédeszköznek bizonyult a felhasználó-központú szemlélet. A visszajelzések számos továbbfejlesztési lehetőségre mutattak rá: a tesztek során gyűjtött kvalitatív adatokon alapulnak mind a rövidtávú javítások és finomhangolások, amelyeket közvetlenül a tesztelés után implementáltam, mind a hosszabb távú, koncepcionális kérdéseket is érintő módosítási javaslatok, amelyeket bemutattam.

Irodalomjegyzék

[1] Kondorosi Károly, Kóczy Annamária et al.: Operációs rendszerek mérnöki megközelítésben, Panem Kiadó Kft., 2000

[2] Dr. Izsó Lajos, Dr. Antalovits Miklós: Bevezetés az információ-ergonómiába, BME Ergonómia és Pszichológia Tanszék, 2000

[3] Jensen Harris: The Story of the Ribbon, in Proceedings of MIX 08, Las Vegas, United States, March 5-7, 2008

[4] Bill Buxton: Sketching User Experiences, Morgan Kaufmann Publishers, 2007

[5] Mitchell Kapor: A Software Design Manifesto, PC Forum, 1990

[6] Robert Elliott, Emilie Herman, John Atkins et al.: The Essential Guide to User Interface Design (Second Edition), Wiley Computer Publishing, 2002

[7] John Bennett, Laura De Young, Bradley Hartfield et al.: Bringing Design to Software, ACM Press, 1996

[8] DigiBarn Computer Museum: The World of Xerox at the DigiBarn [Online], http://www.digibarn.com/collections/xerox-all.html, 2008

[9] Macintosh Human Interface Guidelines, Addison-Wesley Publishing Company, 1992

[10] Steve Krug: Don't Make Me Think, New Riders Press, 2005

[11] Jakob Nielsen: Usability Engineering at a Discount, in Proceedings of the Third International Conference on Human-Computer Interaction, Boston, United States, September, 1989.

[12] Randolph G. Bias, Deborah J. Mayhew et al.: Cost-Justifying Usability, Second Edition: An Update for the Internet Age, Morgan Kaufmann, 2005

[13] Steve Krug: Rocket Surgery Made Easy, New Riders Press, 2009

[14] Jeffrey Rubin, Dana Chisnell: Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests (Second Edition), Wiley Publishing, 2008

[15] Jakob Nielsen: Why You Only Need to Test with 5 Users, Jakob Nielsen's Alertbox, March 19, 2000

[16] Carol M. Barnum: Usability Testing Essentials: Ready, Set...Test!, Morgan Kaufmann, 2010

[17] Gary McLean Hall: Pro WPF and Silverlight MVVM: Effective Application Development with Model-View-ViewModel, Apress, 2010

[18] Thomas Tullis, William Albert: Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics, Morgan Kaufmann, 2008

[19] P. W. Jordan, B. Thomas, B. A. Weerdmeester, A. L. McClelland: Usability Evaluation in Industry, Taylor and Francis, 1996