Migracija MAUI aplikacij z uporabo agentov UI

Alen Granda

Alcad d.o.o., Slovenska Bistrica, Slovenija
alen.granda@alcad.si

Uradni zaton ogrodja Xamarin pred razvojne ekipe postavlja nujnost prehoda na sodobno platformo .NET MAUI. Pri kompleksnih organizacijskih aplikacijah standardna deterministična orodja, kot je Microsoft Upgrade Assistant, pogosto ne zadoščajo. Takšni sistemi namreč vključujejo napredne arhitekturne vzorce, odvisnosti od zunanjih knjižnic (npr. DevExpress) in zahteve po optimizaciji poslovne logike. Prispevek analizira evolucijo migracijskih pristopov z uporabo agentov umetne inteligence (UI) — od lokalnih modelov (Ollama in DeepSeek), ki so omejeni s strojno opremo in velikostjo kontekstnega okna, do oblačnega orodja Claude Code. Na študiji primera produkcijske aplikacije z osmimi projekti in več kot 180.000 vrsticami kode je prikazana metodologija razdelitve migracije na devet logičnih korakov. Rezultati kažejo na 54-odstotni prihranek časa. Dodatno prispevek izpostavi tipične napake agentov ter večplastne varnostne protokole, kot je datoteka .claudeignore, ki skupaj s strokovnim pregledom kode omogočata varno in optimizirano izvedbo tehnološkega prehoda.

Ključne besede:
.NET MAUI
migracija programske opreme
agenti umetne inteligence
inženirstvo navodil
organizacijska varnost podatkov

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-188-3
DOI https://doi.org/10.18690/um.feri.7.2026.17

1 Uvod

Hiter razvoj mobilnih tehnologij organizacijska okolja redno postavlja pred izzive tehnoloških prenov, med katerimi je tudi zaključek življenjskega cikla ogrodja Xamarin. Ker so številne organizacije na tej platformi zgradile robustne, večnivojske sisteme za podporo kritičnim poslovnim procesom, Microsoftova uradna prekinitev podpore zahteva migracijo na sodobno platformo .NET MAUI. V zaprtih podjetniških ekosistemih, kjer se aplikacije navadno nameščajo z uporabo sistemov za upravljanje mobilnih naprav (MDM), je primarni motiv za prehod predvsem zagotavljanje informacijske varnosti, skladnosti z regulatornimi zahtevami ter dolgoročne združljivosti z novimi različicami operacijskih sistemov in strojne opreme.

Migracija kompleksnih organizacijskih rešitev presega zmožnosti standardnih, deterministčnih orodij. Rešitve, kot je Microsoft .NET Upgrade Assistant [1], uspešno izvedejo zgolj osnovne, sintaktične zamenjave imenskih prostorov in formatov projektnih datotek. Vendar povsem odpovejo pri razumevanju kompleksnih arhitekturnih vzorcev (npr. MVVM), odvisnosti od naprednih komercialnih knjižnic tretjih oseb (kot je knjižnica DevExpress [7]) ter pri optimizaciji poslovne logike. Ročni prehod projektov, ki obsegajo več sto tisoč vrstic kode, zahteva nesorazmerno velik vložek inženirskih ur in povečuje tveganje za človeške napake.

Prispevek raziskuje metodologijo migracije aplikacij z uporabo agentov umetne inteligence (UI) kot komplementarnega orodja za avtonomno programiranje. Analiziramo tehnološko evolucijo pristopov — od začetnih poskusov z lokalno izoliranimi modeli, namenjenimi največji možni zaščiti podatkov, do uvedbe oblačnih agentskih sistemov s pomočjo Claude Code [4], ki ponujajo visoko kognitivno zmogljivost in razumevanje celotne topologije programske opreme v realnem času. S študijo primera produkcijske aplikacije z 8 projekti in več kot 180.000 vrsticami kode ovrednotimo korake strukturiranega vodenja agenta, analiziramo tipične napake in anomalije v delovanju modelov ter pokažemo ekonomsko upravičenost in varnostno vzdržnost takšnega simbiotičnega delovnega toka v organizacijskem okolju.

2 Izzivi migracije s platforme Xamarin na .NET MAUI v organizacijskem okolju

Microsoftov uradni zaključek podpore za ogrodje Xamarin predstavlja kritično točko za organizacije, ki upravljajo lastne mobilne rešitve. Čeprav se v organizacijskih okoljih aplikacije distribuirajo interno z uporabo sistemov MDM in niso podvržene pravilom javnih trgovin, kot sta Google Play Store ter Apple App Store, je migracija na .NET MAUI nujna zaradi več kritičnih dejavnikov:

- Varnostna tveganja in skladnost: S koncem podpore ogrodje Xamarin ne prejema več varnostnih popravkov. Odkrite ranljivosti v jedru ogrodja ali njegovih odvisnostih ostajajo odprte, kar povečuje napadalno površino organizacijske infrastrukture in krši interne varnostne standarde ter regulatorne zahteve glede varovanja podatkov.
- Združljivost z novimi različicami OS: Proizvajalci mobilnih naprav, kot so Zebra, Honeywell, Samsung, Apple ipd., redno posodabljajo operacijske sisteme Android in iOS na napravah v uporabi. Zastarela ogrodja ob posodobitvah sistemskih knjižnic pogosto začnejo kazati nepredvideno vedenje, se sesuvajo ali izgubijo dostop do posebnih strojnih funkcij (npr. bralniki črtnih kod).
- Posodobitev strojne opreme: Nakup novih generacij namenskih ali splošnih mobilnih naprav zahteva podporo za najnovejše različice OS, ki jih Xamarin zaradi arhitekturnih omejitev ne more več učinkovito nasloviti.
- Razpoložljivost kadrov in tehnološki dolg: Vzdrževanje sistemov na tehnologiji, ki je zunaj življenjskega cikla, povečuje tehnični dolg organizacije in otežuje pridobivanje ter zadrževanje razvojnih inženirjev, saj se zmanjšuje nabor strokovnjakov, ki želijo delati na opuščenih platformah.

Prehod na platformo .NET MAUI je zato ključen za zagotavljanje dolgoročne stabilnosti, varnosti in tehnološke skladnosti celotnega organizacijskega informacijskega ekosistema.

Kot primarno tehnološko rešitev za prehod Microsoft ponuja orodje .NET Upgrade Assistant [1]. Analiza delovanja tega orodja na kompleksnih projektih kaže, da so njegovi algoritmi omejeni na deterministično sintaktično zamenjavo. Orodje uspešno izvede posodobitev strukture datotek .csproj v novo obliko, preimenuje osnovne imenske prostore (npr. prehod iz Xamarin.Forms v Microsoft.Maui) in posodobi standardne tipe elementov. Vendar je glavna omejitev orodja odsotnost razumevanja širšega konteksta arhitekture. Upgrade Assistant ne zmore:

- Razreševati kompleksnih med-projektnih odvisnosti v večnivojskih rešitvah.
- Samodejno nadomeščati opuščenih programskih vzorcev ali API-jev.
- Avtonomno reševati konfliktov, ki nastanejo zaradi zunanjih knjižnic brez neposredne poti za nadgradnjo.

Posledično izvedba samodejne migracije na obsežnih projektih ustvari veliko število napak pri prevajanju, kar zahteva obsežno ročno posredovanje. To sistemsko vrzel prepoznava tudi razvijalec ogrodja. Microsoft namreč trenutno že sam predlaga in uvaja uporabo umetne inteligence kot ključnega dopolnilnega orodja za uspešno izvedbo migracij projektov iz platforme Xamarin na .NET MAUI [1].

Identificirani izzivi potrjujejo, da migracija kompleksnih organizacijskih sistemov presega okvire enostavnega sintaktičnega prevajanja programske kode. Kombinacija neizogibnega tehnološkega preloma, arhitekturne zapletenosti, odvisnosti od specializiranih knjižnic (kot je DevExpress) ter potrebe po hkratni optimizaciji poslovne logike ustvarja vrzel, ki je tradicionalna deterministična orodja ne morejo premostiti. Ker ročna migracija projektov takšnega obsega zahteva nesorazmerno veliko časovnih in kadrovskih virov, se implementacija naprednih kognitivnih orodij ponuja kot logična rešitev. V nadaljevanju prispevka zato analiziramo evolucijo uporabe sistemov umetne inteligence — od začetnih poskusov z lokalnimi modeli do uvedbe visokozmogljivih oblačnih agentov, prilagojenih za avtonomno programiranje.

3 Od lokalnih modelov do oblačnih agentov UI

Začetna faza raziskovanja je bila usmerjena v vzpostavitev lokalnega okolja za izvajanje modelov umetne inteligence. Glavni motiv za ta pristop je izviral iz organizacijskih zahtev po varovanju intelektualne lastnine, preprečevanju odtekanja izvorne kode zunaj lokalnega omrežja ter potrebi po popolnem nadzoru nad podatkovnimi tokovi.

Tehnična izvedba je temeljila na odprtokodnem ogrodju Ollama [2], ki omogoča lokalno izvajanje velikih jezikovnih modelov z visoko stopnjo optimizacije porabe pomnilnika. Kot osrednje kognitivno jedro so bili izbrani modeli iz družine DeepSeek [3].

Modeli DeepSeek so v skupnosti razvojnih inženirjev prepoznani kot vodilni na področju avtonomnega programiranja in analize kompleksnih odvisnosti v okolju .NET. Ponujajo napredno razumevanje specifične sintakse jezika C# in struktur ogrodja .NET, pri čemer na standardiziranih testih programiranja (npr. HumanEval) dosegajo rezultate, primerljive ali celo boljše od komercialnih zaprtih rešitev [3].

Kljub izpolnitvi varnostnih kriterijev je lokalni pristop pri implementaciji na rešitvi z 8 projekti naletel na kritične arhitekturne in strojne omejitve. Za testiranje smo uporabili model DeepSeek-R1:14B na inženirski delovni postaji z naslednjo specifikacijo:

- Procesor**:** AMD Ryzen 7 5800X 8-Core Processor (3.8 GHz)
- Sistemski pomnilnik: 32 GB RAM
- Grafična kartica: NVIDIA T400 (2 GB VRAM)

Ob zagonu te konfiguracije se je najbolj očitno pokazalo ozko grlo v zmogljivosti strojne opreme. Model s 14 milijardami parametrov za optimalno delovanje v realnem času zahteva znatno več grafičnega pomnilnika, kot ga ponuja kartica NVIDIA T400 z 2 GB VRAM. Zaradi pomanjkanja VRAM-a je bil sistem prisiljen naložiti večino modela v sistemski pomnilnik. Hitrost generiranja žetonov je upadla, izvajanje kompleksnejših operacij sklepanja je postalo časovno nesprejemljivo za interaktivno delo.

Ugotovljene omejitve lokalnih modelov so narekovale ponovno ovrednotenje strategije in prehod na namenska oblačna orodja za aktivno programiranje. Kot osrednja tehnološka rešitev je bilo izbrano orodje Claude Code podjetja Anthropic [4]. Za razliko od lokalnih rešitev je Claude Code sposoben sočasno analizirati celotno topologijo organizacijske aplikacije. Agent v realnem času prepoznava povezave med slojem uporabniškega vmesnika, poslovno logiko in servisnim slojem. Ta sposobnost omogoča celostno migracijo, pri kateri agent ob posodobitvi določenega servisa samodejno identificira in posodobi vse odvisne komponente v preostalih projektih, s čimer se občutno zmanjša število napak v fazi gradnje programske kode.

4 Metodologija priprav in vodenja agenta UI

Uporaba oblačnih agentov UI, kot je Claude Code, v zapletenih migracijskih procesih zahteva prehod od klasičnega pisanja pozivov k sistematičnemu inženirstvu programskih navodil. Da bi agent deloval avtonomno in v skladu z organizacijskimi standardi, smo vzpostavili voden delovni tok (angl. workflow), pri katerem človek deluje kot arhitekt in nadzornik, stroj pa kot izvajalec manjših nalog.

Preden se delo prepusti agentu UI, je ključnega pomena vzpostavitev izhodiščnega vzorca kodiranja. V ta namen smo najprej ročno migrirali eno od aplikacij, ki je bila reprezentativni vzorec migracije. S tem smo ustvarili referenčni projekt, ki je agentu služil kot empirični dokaz želene ciljne strukture. Agent je na podlagi primerjave stanja reprezentativnega projekta pred migracijo in po njej natančno razbral semantiko pretvorbe.

Dodatno smo uveljavili še vodnik MAUI_MIGRATION_GUIDE.md. Vsebuje natančna pravila o poimenovanju imenskih prostorov, pravila za branje odvisnosti ter standarde za ravnanje z zunanjimi knjižnicami. Med samo izvedbo migracije smo uporabili način načrtovanja (ang. Plan Mode). To pomeni, da je moral agent pred vsakim urejanjem datotečnega sistema najprej analizirati obstoječo kodo, pripraviti taksonomijo sprememb in jo predstaviti arhitektu v potrditev. Šele po eksplicitni odobritvi načrta je lahko agent prešel v fazo izvajanja.

Zaradi obsega aplikacije je bila migracija razdeljena na zaporedne, logično zaključene mikronaloge. Vsak korak je bil poslan agentu kot izoliran ukaz z natančno definiranimi zahtevami (Slika 1):

1. Priprava projektne strukture: Prva mikronaloga obsega inicializacijo nove projektne topologije na podlagi vnaprej pripravljene predloge. Agent analizira izvorne datoteke predloge, določi konvencije poimenovanja ter ustvari ciljne podprojekte. Sledi samodejni prenos infrastrukturnih datotek, rekurzivno preimenovanje map ter posodobitev datotek MauiProgram.cs in AppShell.xaml.
2. Uvedba orodja Refitter [8]: Pomembna arhitekturna sprememba v primerjavi s prvotno aplikacijo Xamarin je posodobitev komunikacijskega sloja z zalednimi sistemi. V prejšnji arhitekturi se je za generiranje odjemalcev uporabljalo orodje NSwagStudio [5], ki je generiralo obsežne in toge monolitne datoteke. V novi arhitekturi smo to zamenjali s sodobnim orodjem Refitter [8]. Ta korak je bil izveden ročno, za postavitev trdnih temeljev pred predajo dela agentu UI.
3. Migracija lokalizacije: Agent identificira stare .resx datoteke, jih preseli v centraliziran projekt ter izvede ekstrakcijo trdo kodiranih nizov iz datotek pogleda XAML in logike C# v slovenske in angleške vire ter ob tem implementira osrednjega skrbnika LocalizationResourceManager, ki skrbi za podporo več jezikom hkrati.

4. Migracija domenskih modelov**:** Na podlagi predhodno pripravljenih podatkovnih prenosnih objektov (DTO) iz orodja Refitter v 2. koraku agent samodejno ustvari ustrezne domenske modele. Hkrati zgradi profile za knjižnico AutoMapper, pri čemer dosledno uporablja deklaracije metode CreateMap.

Slika 1: Diagram postopka migracije aplikacije iz ogrodja Xamarin.Forms v ogrodje .NET MAUI.

5. Prenova storitvenega sloja: V tem koraku agent izvede preoblikovanje storitvene logike, pri katerem strogo loči vmesnike od njihovih implementacij. Stari klici, generirani z orodjem NSwag, se nadomestijo s sodobnimi odjemalci Refitter. Agent znotraj storitev implementira IMapper za pretvorbo DTO-jev v domenske modele ter samodejno registrira storitve. Pomembno pravilo, ki ga agent tukaj upošteva, je prepoved neposrednega klicanja odjemalcev HTTP iz modelov pogleda. Klici se morajo izvajati izključno z uporabo metod poslovnih storitev na podlagi generične metode SendRequest().

6. Modernizacija modelov pogleda in prenova navigacije: V tej fazi agent prenovi sloj predstavitvene logike z uporabo ogrodja CommunityToolkit.Mvvm [6]. Odstrani se obsežna koda za ročno proženje dogodkov sprememb lastnosti, ki jo nadomestijo izvorno generirane lastnosti in ukazi na podlagi atributov

[ObservableProperty] in [RelayCommand]. Prejšnji način navigacije Navigation.PushAsync se pretvori v vzorec lupine ogrodja MAUI Shell.Current.GoToAsync(). Hkrati se uveljavi pridobivanje storitev iz konstruktorja.

7. Abstrakcija in transformacija podatkovnih tabel: Ker so aplikacije v naši organizaciji močno odvisne od prikazov podatkov, je bil v reprezentativnem vzorcu aplikacije razvit generičen pogled za prikaz podatkov v obliki tabele, imenovan DynamicDataGridContentView. Naloga agenta je bila, da analizira stare datoteke uporabniških vmesnikov XAML, identificira implementacije tabel ter generira prečiščen, dinamičen pogled, ki stolpce generira programsko z uporabo metode GenerateColumns(). Definicije stolpcev za specifične tipe modelov agent strukturira znotraj statičnega razreda DataGridHelper. Ker je takšnih tabel veliko, smo ta korak izolirali in ga poslali agentu kot samostojen ukaz.
8. Transformacija uporabniškega vmesnika in integracija knjižnice DevExpress: Sledi celovita prenova datotek uporabniškega vmesnika XAML. Agent samodejno posodobi imenske prostore iz Xamarin.Forms v Microsoft.Maui.Controls ter integrira gradnike knjižnice DevExpress. Iz ozadja se odstrani vsa poslovna logika, dogodki elementov se povežejo neposredno na ukaze modela pogleda.
9. Dopolnitve videza aplikacije: Zadnja faza obsega vizualno dopolnjevanje in prenos naprednih vizualnih funkcionalnosti. Agent znotraj datoteke Styles.xaml poenoti videz vizualnih komponent. Agent implementira podporo za podrsanje elementov z uporabo ikon. Ključni del te faze je uspešna obnova stare logike CustomCellStyleCommands, ki je skrbela za dinamično barvanje celic tabel, ter njena uspešna prilagoditev za ekosistem ogrodja .NET MAUI. Na koncu agent poenoti videz pojavnih oken in uredi poravnavo akcijskih gumbov.

Po vsakem zaključenem modulu oziroma koraku je agent dolžan zagnati ukaz za prevajanje rešitve. Če prevajalnik sporoči napake, se agent vrne v način načrtovanja, analizira izpis napak ter samostojno uporabi popravke. Arhitekt prevzame nadzor nad kodo šele takrat, ko je posamezni modul uspešno preveden brez napak.

5 Varnostni protokoli in zaščita intelektualne lastnine v oblaku

Prehod z lokalnih modelov na visoko zmogljive oblačne agente, kot je Claude Code, izpostavi organizacijo specifičnim varnostnim tveganjem. Glavni izziv v organizacijskem okolju je zaščita integritete izvorne kode in preprečevanje odtekanja intelektualne lastnine na zunanje strežnike ponudnikov storitev umetne inteligence. Za omilitev tveganj in nadzor nad podatkovnimi tokovi je bil vzpostavljen hibridni varnostni model, ki temelji na dveh tehničnih stebrih:

- Selektivna izolacija podatkov z uporabo datoteke .claudeignore: Podobno kot datoteka .gitignore v sistemih za upravljanje različic, datoteka .claudeignore omogoča izrecno določitev datotek in map, do katerih oblačni agent pod nobenim pogojem nima dostopa. V to konfiguracijo so bili sistemsko vključeni: vsi produkcijski konfiguracijski podatki, certifikati in šifrirni ključi (.env, appsettings.json), mape gradnje (bin/ in obj/) ter posebni moduli, ki vsebujejo občutljive podatke o poslovnih procesih podjetja.
- Človeški nadzor: Uveden je bil strog protokol strokovnega pregleda. To pomeni preverjanje, ali je agent spreminjal izključno datoteke v odobrenem obsegu in ali ni prišlo do nenamernega vnašanja varnostnih ranljivosti (npr. trdo kodiranih dostopov ali nepreverjenih zunanjih odvisnosti) ter ocenjevanje, ali generirana koda dosledno sledi vzorcem, definiranim v izhodiščnem vodniku MAUI_MIGRATION_GUIDE.md. Šele po uspešnem pregledu in lokalnem testu delovanja se programska koda odobri za združitev v glavno razvojno vejo.

Simbiotični odnos med arhitektom in agentom UI tako zagotavlja, da ima človek popolno in zadnjo besedo nad vsemi spremembami, kar zmanjšuje tveganja oblačne obdelave podatkov.

6 Analiza rezultatov

Za objektivno oceno predlagane metodologije smo izvedli primerjavo med konvencionalnim ročnim postopkom migracije in agentsko podprtim delovnim tokom. Primarni cilj analize je bil ovrednotiti časovne prihranke, identificirati tipične vzorce napak sistemov umetne inteligence ter ovrednotiti ekonomsko upravičenost uvedbe naprednih agentov v razvojni cikel.

Analiza je bila izvedena na produkcijski mobilni aplikaciji, ki se uporablja v internem logističnega in proizvodnega ekosistema. Aplikacijo predstavljajo naslednji parametri:

- Obseg rešitve**:** 8 ločenih podprojektov znotraj enotne rešitve.
- Uporabniški vmesnik**:** 12 uporabniških pogledov XAML z gostimi podatkovnimi strukturami in interaktivnimi elementi.
- Velikost kodne baze**:** več kot 180.000 vrstic kode.

Za potrebe študije primera je bila izvedena primerjava med časovnim okvirjem za popolno ročno migracijo in izvedbo s pomočjo orodja Claude Code. Ročna migracija je bila izvedena pred uporabo agenta UI. Rezultate prikazuje Tabela 1.[i]

Tabela 1: Primerjava ročne in agentske migracije.

Metrika	Ročna migracija reprezentativnega projekta	Agentsko podprta migracija
Čas izvedbe	3 tedni (120 ur)	9 dni (56 ur)
Obseg rešitve	8 projektov	8 projektov
SLOC	171 000	180 000
Potrebni ročni popravki (Commits)	/	~ 30
Stabilnost prve gradnje	Nizka (več 100 napak)	Visoka (iterativno odpravljanje z uporabo sprotne gradnje)

Rezultati kažejo na precejšnje zmanjšanje potrebnega časa za izvedbo celotne migracije. Ročni prehod celotne rešitve z danimi omejitvami in funkcionalnostmi bi od izkušenega inženirja zahteval približno tri tedne intenzivnega dela (vključno z izvedbo migracije, testiranjem in prenosom v produkcijsko okolje). Z uporabo orodja Claude Code in organiziranim pristopom je bil proces zaključen v dobrem tednu dni, kar predstavlja 54-odstotni prihranek časa.

Med procesom migracije smo sistematično beležili primere, ko je agent odpovedal ali generiral napačne programske vzorce (t. i. halucinacije). Te anomalije razdelimo v pet glavnih kategorij:

- Omejitve pri interpretaciji zaprtih zunanjih odvisnosti: agent ni znal implementirati in prenesti logike, povezane z namensko knjižnico HubService. Ker gre za interno, lokalno knjižnico NuGet, ki ni del javno dostopnih podatkov za učenje modela, agent brez zunanje dokumentacije ni mogel pravilno razbrati njene semantike in je pustil komentar z opisom opravila (Slika 2).
- Pomanjkljivo in napačno preslikovanje podatkovnih modelov: pri ročnem preslikovanju modelov je agent občasno izpustil določene lastnosti ali pa jih je preslikal nepravilno (Slika 3, Slika 4).
- Ignoriranje referenčnih primerov: kljub eksplicitnim navodilom in obstoju referenčnega projekta je agent pri transformaciji tabel večkrat poskušal uporabiti zastareli koncept iz ogrodja XAML. Ni uspel takoj prepoznati, da nova arhitektura zahteva uporabo prenovljene komponente DynamicDataGridContentView, ki na tipiziran način dinamično generira uporabniški vmesnik.
- Izpad kompleksnih kontrol (CustomCellStyleCommand): zaradi sprememb v načinu definiranja vizualnih stanj v ogrodju .NET MAUI je agent na več mestih izpustil implementacijo ukaza

CustomCellStyleCommand (Slika 5). Kompleksne logike barvanja celic na podlagi pogojev ni uspel avtonomno prilagoditi novemu ekosistemu, zato je bil na tem delu potreben ročni poseg arhitekta.

– Težave pri definiranju vizualne identitete in videza aplikacije: Agent je imel težave pri zagotavljanju enakega ali izboljšanega videza uporabniškega vmesnika v primerjavi z izvirno različico. Brez dodatnega stilskega usmerjanja je generiral osnovne, neoptimizirane postavitve elementov, napačne odmike ter nedosledno tipografijo. Vizualni rezultat neposredno po zaključku avtonomnega dela agenta in pred kakršnimi koli ročnimi popravki razvijalca prikazuje slika 6.

Vse navedene napake so bile uspešno odpravljene s kombinacijo natančnejšega usmerjanja agenta v načinu načrtovanja v zadnjem koraku in končnih ročnih popravkov razvijalca.

```
        2 references
42      partial void OnWorkOrderChanged(WorkOrderModel value)
43      {
44          if (value == null) return;
45          _screenService.SetScreenAlwaysOn();
46          Title = string.Format(
47              LocalizationResourceManager.Instance["BatchPreparationPageTitle"]?.ToString() ?? "",
48              value.ProductionLineName, value.BatchId, value.AlloyName, value.WorkOrderId);
49          IsSummaryVisible = AppSettingsService.AppSettings.TenantId == "27";
50          Task.Run(async () =>
51          {
52              await FetchBatchSuggestionDataAsync();
53              await FetchScannedDataAsync();
54          });
55          // TODO: IHubService – RegisterHubListeners()
56      }
57
```

Slika 2: Komentar agenta UI za nadaljnje opravilo.

```
47  public string GenerateQrCodeStringForMobile(UserInfoModel user, WorkOr
48  {
49      var model = new QrCodeForMobileModel
50      {
51          Id = user.Id,
52          Login = user.Login,
53          FullName = user.FullName,
54          Rfid = user.RFID,
55          Token = user.Token,
56          LdapUid = user.LdapUid,
57          BatchPreparationInfo = new BatchPreparationInfoModel
58          {
59              BatchId = workOrder.BatchId,
60              ProductionLineId = workOrder.ProductionLineId,
61              ProductionLineName = workOrder.ProductionLineName,
62              ProductionLineCode = string.Empty,
63              ForkliftLocationId = forkliftLocationId,
64              ForkliftLocationCode = forkliftLocationCode,
65-             QrCodeType = isBatchAlloying ? "alloying" : "casting"
66          }
67      };
68
69      var json = JsonSerializer.Serialize(model);
70      return Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(j
71  }
```

```
47  public string GenerateQrCodeStringForMobile(UserInfoModel user, WorkOrderModel workOrder
48  {
49      var model = new QrCodeForMobileModel
50      {
51          Id = user.Id,
52          Login = user.Login,
53+         FirstName = user.FirstName,
54+         LastName = user.LastName,
55+         Email = user.Email,
56          FullName = user.FullName,
57          Rfid = user.RFID,
58          Token = user.Token,
59          LdapUid = user.LdapUid,
60          BatchPreparationInfo = new BatchPreparationInfoModel
61          {
62              BatchId = workOrder.BatchId,
63              ProductionLineId = workOrder.ProductionLineId,
64              ProductionLineName = workOrder.ProductionLineName,
65              ProductionLineCode = string.Empty,
66              ForkliftLocationId = forkliftLocationId,
67              ForkliftLocationCode = forkliftLocationCode,
68+             QrCodeType = isBatchAlloying ? "BatchAlloying" : "BatchComposition"
69          }
70      };
71
72      var json = JsonSerializer.Serialize(model);
73      return Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(json));
74  }
```

Slika 3: Manjkajoče preslikave po kodiranju agenta UI.

```
95  private async Task FetchScannedDataAsync()
96  {
97      if (WorkOrder == null) return;
98      IsScannedDataRefreshing = true;
99      var stock = await _compositionProposalService.GetPackagesInDeliveryAsync(WorkOrder.BatchId);
100     if (stock != null)
101     {
102         await MainThread.InvokeOnMainThreadAsync(() =>
103         {
104             ScannedData = stock.AlloyList == null
105                 ? new BindingList<AlloyWithLocationAndQuantityModel>()
106                 : new BindingList<AlloyWithLocationAndQuantityModel>(stock.AlloyList.ToList());
107
108-            _scannedTransportUnitIds = stock.TransportUnits?.Select(x => x.PisZapStev).ToList()
109                                        ?? new List<int>();
110
111             OstanekQuantity = stock.LeftoverQuantityFromPrevBatches;
112             SkeniranoQuantity = stock.AlloyList?.Sum(x => x.Quantity) ?? 0m;
113             SkupnoQuantity = OstanekQuantity + SkeniranoQuantity;
114         });
115     }
116     IsScannedDataRefreshing = false;
```

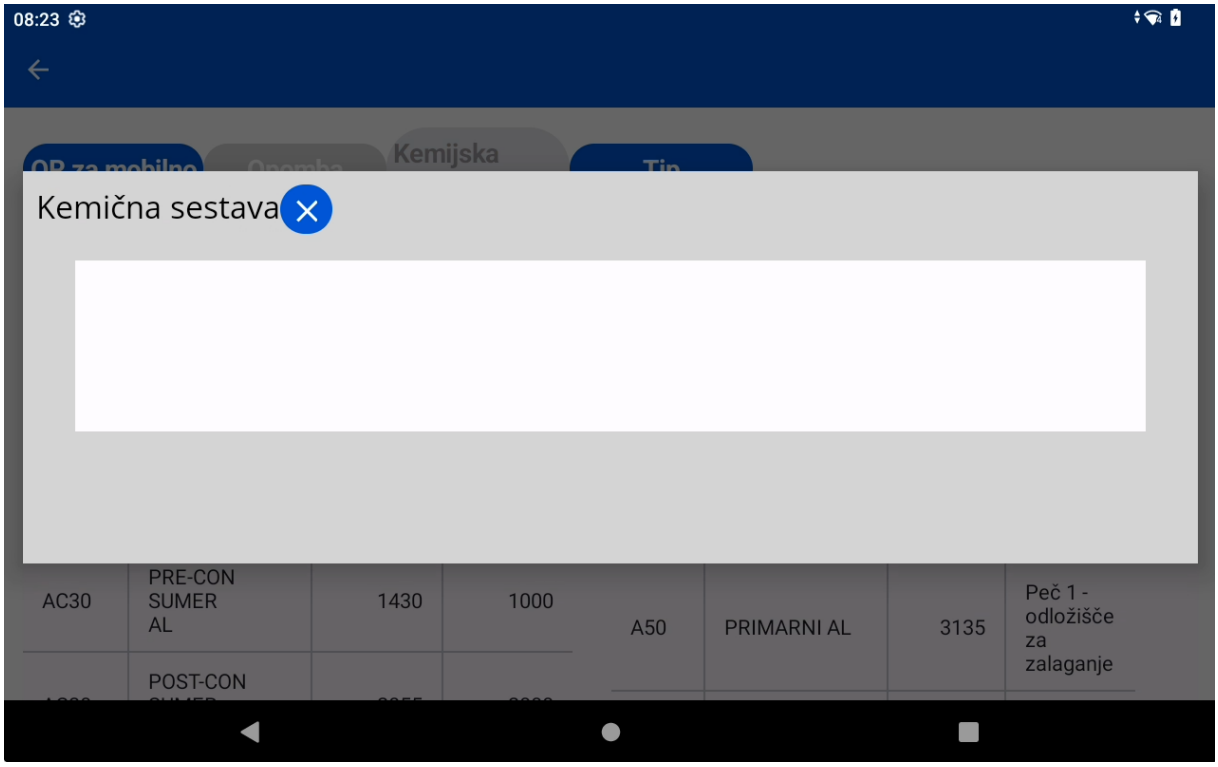
```
95  private async Task FetchScannedDataAsync()
96  {
97      if (WorkOrder == null) return;
98      IsScannedDataRefreshing = true;
99      var stock = await _compositionProposalService.GetPackagesInDeliveryAsync(WorkOrder.BatchId);
100     if (stock != null)
101     {
102         await MainThread.InvokeOnMainThreadAsync(() =>
103         {
104             ScannedData = stock.AlloyList == null
105                 ? new BindingList<AlloyWithLocationAndQuantityModel>()
106                 : new BindingList<AlloyWithLocationAndQuantityModel>(stock.AlloyList.ToList());
107
108+            _scannedTransportUnitIds = stock.TransportUnits?.Select(x => x.TransportUnitId).ToList()
109                                        ?? new List<int>();
110
111             OstanekQuantity = stock.LeftoverQuantityFromPrevBatches;
112             SkeniranoQuantity = stock.AlloyList?.Sum(x => x.Quantity) ?? 0m;
113             SkupnoQuantity = OstanekQuantity + SkeniranoQuantity;
114         });
115     }
116     IsScannedDataRefreshing = false;
```

Slika 4: Primer nepravilnega preslikovanja lastnosti.

```
169     [RelayCommand]
170-    private void CustomCellStyleForRecommendedPackages(object args) { }
171
172     [RelayCommand]
173-    private void CustomCellStyleForScannedPackages(object args) { }
174
```

```
169     [RelayCommand]
170+    private void CustomCellStyleForRecommendedPackages(object args)
171+    {
172+        if (args is DevExpress.Maui.DataGrid.CustomCellAppearanceEventArgs a && a.Item is BatchCompositionProposalLineModel g)
173+        {
174+            if (g.QuantityInDelivery <= 0)
175+            {
176+                a.BackgroundColor = Colors.SkyBlue;
177+            }
178+            else
179+            {
180+                a.BackgroundColor = g.QuantityInDelivery < g.QuantitySuggested ? Colors.LightSalmon : Colors.LightGreen;
181+            }
182+        }
183+    }
184
185     [RelayCommand]
186+    private void CustomCellStyleForScannedPackages(object args)
187+    {
188+        if (args is DevExpress.Maui.DataGrid.CustomCellAppearanceEventArgs a && a.Item is AlloyingQuantityModel p)
189+        {
190+            if (p.Quantity > 0 && p.InappropriateAccordingToBatchCompositionProposal)
191+            {
192+                // if it is alloying element (Klasifi2 = 98), color it yellow
193+                if (p.AlloyTypeId == "98")
194+                {
195+                    a.FontColor = Color.FromRgb(220, 157, 0);
196+                }
197+                else
198+                {
199+                    a.FontColor = Colors.Tomato;
200+                }
201+            }
202+        }
203+    }
204
```

Slika 5: Manjkajoče implementacije metod stiliranja celic tabel.



Slika 6: Primer videza aplikacije po migraciji z agentom UI brez ročnih popravkov.

7 Zaključek

Rezultati prispevka in praktične implementacije agentsko podprte migracije potrjujejo, da sistemi umetne inteligence spreminjajo procese vzdrževanja in posodabljanja obsežnih programskih rešitev. Prehod z deterministčnih orodij na kognitivne agente omogoča, da se proces migracije iz gole sintaktične zamenjave preoblikuje v nadzorovano migracijo programske kode v realnem času.

Empirični podatki iz študije primera kažejo, da se je čas izvedbe migracije zmanjšal za 54 % — s treh tednov ročnega dela na dober teden inženirskega časa. Kljub širokemu kontekstnemu oknu in kognitivni globini oblačnih agentov, kot je Claude Code, njihovo delovanje izkazuje določene omejitve. Anomalije, kot so pomanjkljivo preslikovanje modelov, nerazumevanje lokalnih knjižnic ter nagnjenost k uporabi zastarelih komponent pogleda namesto prenovljenih predlog, dokazujejo, da popolna avtomatizacija teh procesov trenutno še ni izvedljiva.

Uspešna, varna in ekonomsko upravičena migracija zato ne temelji na polni avtonomiji stroja, temveč na simbiozi med programskim arhitektom in agentom UI. Človek v tem procesu prevzema vlogo postavljanja izhodiščnih vzorcev, priprave sistemskih navodil ter večplastne zaščite intelektualne lastnine z uporabo konfiguracij .claudeignore in arhitekturne anonimizacije. Končni korak strokovnega pregleda kode zagotavlja, da razvijalec ohranja popoln nadzor nad produkcijskim okoljem.

Uporaba opisanega pristopa nakazuje, da je z ustreznim inženirstvom navodil in implementacijo strogih varnostnih protokolov mogoče obvladovati tveganja, povezana z integracijo oblačnih modelov umetne inteligence. To organizacijam omogoča, da proces migracije na novejše različice ogrodij izvedejo z višjo stopnjo predvidljivosti in optimizacije razvojnega cikla.

Literatura

[1] https://learn.microsoft.com/en-us/dotnet/core/porting/upgrade-assistant-overview, What is .NET Upgrade Assistant?, obiskano 29. 6. 2026.

[2] https://ollama.com/, Ollama, obiskano 29. 6. 2026.

[3] Guo, D. et al. (DeepSeek-AI). *DeepSeek-Coder: Touching the Ceiling of Open Source Code Intelligence*. arXiv preprint.

[4] https://claude.com/product/claude-code, Claude Code, obiskano 29. 6. 2026.

[5] https://github.com/RicoSuter/NSwag/wiki/nswagstudio, NSwagStudio, obiskano 29. 6. 2026.

[6] https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/, Introduction to the MVVM Toolkit, obiskano 29. 6. 2026.

[7] https://www.devexpress.com/maui/, .NET MAUI Controls, obiskano 29. 6. 2026.

[8] https://refitter.github.io/, Refitter, obiskano 30. 6. 2026.