Proiectarea Sistemelor Software Complexe

Curs 9 –Linii de Produse Software

9.1 Context

Reutilizarea resurselor software este un concept care în ultimii 20 de ani a devenit din ce in ce mai popular. Ideal, dezvoltatorii de produse software ar trebui să poată crea sisteme software de înaltă calitate prin asamblarea de componente software existente. În trecut accentul s-a pus mai mult pe reutilizarea (la scară mică) unor funcții individuale, a librăriilor de funcții și tipuri de date și a tehnologiilor independente de domeniu. Astfel de abordări s-au dovedit a fi benefice, dar nu exploatează la maxim conceptul de reutilizare software.

Reutilizarea resurselor software este ușor de aplicat atunci când componentele software sunt cunoscute foarte bine și realizează exact ceea ce este nevoie. Dar o componentă care face "aproape ceea ce trebuie" este de ce cele mai multe ori complet inutilă. Abordările moderne în ceea ce privește reutilizarea resurselor software, precum Liniile de Produse Software (Software Product Lines - SPL), oferă suport pentru reutilizarea la scară largă. Dezvoltarea bazată pe Liniile de Produse Software s-a dovedit a fi un mod eficient de a beneficia de reutilizarea și variația resurselor software. Acest tip de dezvoltare dacă este folosit corect duce la reducerea costurilor, la reducerea timpului de dezvoltarea și la creșterea productivității.

Dezvoltarea bazată pe SPL constă din dezvoltarea produselor software similare prin combinarea componentelor software comune cu cele specifice fiecărui produs, obținându-se astfel o variație a funcționalității oferite de componentele centrale. În Fig. 9.1 sunt prezentate schematic două produse de tipul calculator , amândouă folosind ca și motor de calcule o componentă care implementează funcționalitatea unui calculator generic. Funcționalitatea care diferă este implementată de componente separate, de exemplu cele două calculatoare folosesc butoane diferite.

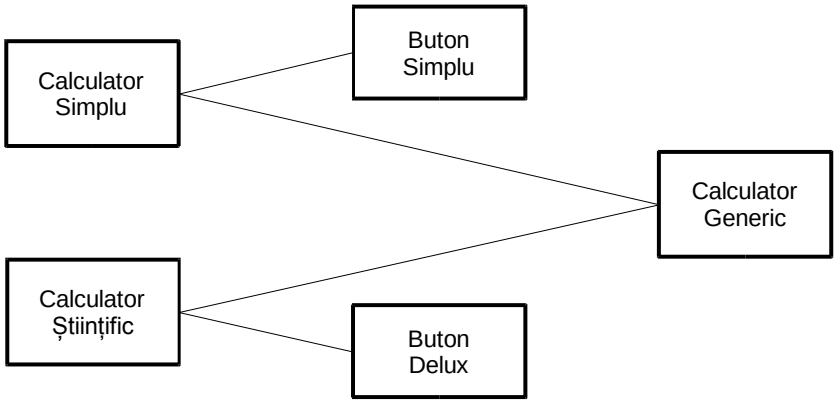

Fig. 9.1. Exemplu de linie de produse software.

Pentru orice produs dintr-o linie de produse software, aproape toată funcționalitatea este implementată prin refolosirea de componente de bază. Aceste componente de bază implementează funcționalitate de bază care este uniformă pentru toate produsele din SPL și oferă suport pentru funcționalități care variază și care pot fi selectate de către fiecare produs. Punctele de variație ale componentelor de bază oferă o interfață prin care se poate alege funcționalitatea care variază. Pentru fiecare produs sunt instanțiate anumite puncte de variație ale componentelor de baza, în plus se pot dezvolta noi componente specifice produsului.

Variația software are mai multe roluri atunci când se folosește dezvoltarea bazată pe SPL. Cel mai evident rol este acela de a oferi suport pentru diferențele de funcționalitate între produsele unui SPL. Variația software poate fi folosită și pentru a oferi suport pentru diferențe non-funcționale (scalabilitate, performanță sau securitate). Un alt rol mai puțin recunoscut este acela de a ajuta la rezolvarea conflictelor între modificările cerute pentru componentele de bază.

SPL nu este doar o problemă de arhitectură, design sau programare. SPL influențează întregul ciclu de viață al procesului de dezvoltare software.

9.2 Beneficiile Dezvoltării SPL

Un SPL este util atunci când se dorește dezvoltarea unui set de produse care partajează funcționalități de bază. De obicei un SPL țintește o piață largă, iar fiecare produs din SPL țintește un anumit segment de piață. Unele SPL-uri au drept scop dezvoltarea unor variante ale unui produs standard, fiecare varianta fiind destinată unui anumit client.

Gama de produse a unei linii de produse este reprezentată de toate variațiile suportate de componentele care formează nucleul SPL-ului. Produsele dintr-un SPL vor fi cu siguranță în gama SPL-ului, dar componentele personalizate vor oferi posibilitatea dezvoltării de produse care nu fac parte din gama SPL-ului. Pentru a maximiza beneficiul unui SPL trebuie ca, gama SPL-ului să cuprindă atât piața de interes a companiei (permițând astfel dezvoltarea rapidă și eficientă de noi produse pentru piața de interes), cât și toate funcționalitățile necesare pentru produse dezvoltate de companie.

Cel mai evident beneficiu al unui SPL este reprezentat de creșterea productivității. Costurile aferente dezvoltării și întreținerii componentelor de bază care formează nucleul SPL-ului nu se însumează pentru fiecare produs din SPL ci se împart. Compania poate beneficia de dezvoltarea unui număr mare de produse. Un SPL este scalabil în ceea ce privește numărul de produse întrucât costul marginal al adăugării unui nou produs este minim.

Un SPL bine proiectat are și alte avantaje. De exemplu dacă componentele care formează nucleul SPL-ului sunt bine definite, atunci timpul necesar dezvoltării unui nou produs este mult mai mic decât în cazul nu care nu se folosește dezvoltarea bazată pe SPL. Acest lucru se datorează în special faptului că nu este necesară implementarea funcționalității care formează nucleul SPL-ului, ci este necesară doar implementarea funcționalității noi (unice).

Dezvoltarea bazată pe SPL contribuie și la creșterea calității. În dezvoltarea tradițională același defect se poate să fie în mai multe produse, el trebuind să fie reparat în fiecare produs, dar în cazul unui SPL un defect care apare la nivelul nucleului este reparat o singură dată. Mai mult chiar dacă defectul a fost găsit pentru un singur produs, toate celelalte produse care folosesc funcționalitatea în care a fost găsit defectul vor beneficia de pe urma remedierii defecţiunii.

Alte beneficii secundare ale utilizării SPL-ului sunt:

- facilitează actualizarea produselor la o nouă versiune a nucleului;

- identificarea componentelor de bază se face mult mai ușor;
- identificarea diferențelor dintre produse este de asemenea ușor de realizat;
- facilitează identificarea funcționalității care va fi adăugată în viitor în SPL.

9.3 Arhitectura Unei Linii de Produse

Dezvoltarea bazată pe SPL folosește Arhitectura bazată pe Linii de Produse (Product Line Architecture - PLA). O arhitectura PLA asigură reutilizarea componentelor de baza ale unui SPL. Obiectivele unei arhitecturi PLA în ceea ce privește reutilizarea și variația sunt:

- suport sistematic pentru funcționalitate variată pre-planificată;
- posibilitatea produselor din SPL să aleagă cu ușurință opțiunile din funcționalitatea variată.

O arhitectură PLA îndeplinește aceste obiective utilizând o varietate de mecanisme tehnice care permit reutilizarea și varietatea.

9.3.1 Mecanisme Care Asigură Reutilizarea

Pentru a putea reutiliza componentele software dezvoltatorii software trebuie să:

- să găsească și să înțeleagă componentele software;
- să încorporeze componentele software în propriul context de dezvoltare;
- să folosească componentele invocând funcționalitatea acestora.

Găsirea și localizarea componentelor software. Inginerii software folosesc documentație API și manualele pentru a permite reutilizarea librăriilor software. Pentru dezvoltarea bazată pe SPL se folosește documentarea procedurilor de utilizare și creare a instanțelor componentelor software care reprezintă nucleul SPL-ului.

Introducerea componentelor software în contextul de dezvoltare. După ce o componentă software a fost găsită un programator trebuie să o facă disponibilă pentru a putea fi utilizată. Există mai multe modalități prin care o componentă poate fi adusă într-un context de dezvoltare. Aceste modalități pot fi împărțite în categorii în funcție de momentul de timp când se realizează legătura cu componentele software reutilizabile. Principalele momente de timp la care se poate realiza legătura cu o anumită componentă sunt:

- în timpul programării – utilizând controlul versiunii pentru codul sursă;
- în timpul compilării – utilizând controlul versiunii pentru librării statice;
- în timpul linkeditării – se realizează pentru librăriile dinamice de către sistemul de operare sau mașina virtuală;
- în timpul rulării – se realizează de către tehnologii middleware sau prin mecanisme specifice aplicației de configurare și plug-in-uri dinamice.

Realizarea legăturii la momente de timp timpurii (în timpul programării sau compilării), facilitează utilizarea variației ad-hoc. Realizarea legăturii la momente de timp târzii (linkeditare sau în timpul rulării), întârzie legarea de o anumită soluție si facilitează obținerea de avantaje de pe urma variației sistematice. Arhitecturile PLA mature ale unui SPL tind să folosească legăturile târzii.

Invocarea componentelor software. Pentru a permite invocarea componentelor software limbajele de programare oferă mecanisme de apelare de tipul: procedură, funcție și metodă. Pentru sistemele software distribuite bazate pe standarde precum CORBA sau SOAP, sunt disponibile mecanisme care permit programatorilor să apeleze componente software care rulează pe alte mașini. Mecanismele de

invocare disponibile pentru un SPL sunt cele disponibile în cazul dezvoltării clasice a unui sistem software.

9.3.2 Gestionarea Configurației Software Pentru Reutilizare

Cele mai utilizate metode de legare a componentelor reutilizabile sunt cele din timpul programării și a compilării. Acest lucru face ca, gestionarea configurației software (Software Configuration Management - SCM) să fie un proces critic pentru un SPL. SCM constă din controlul versiunii și controlul modificărilor.

SCM este mult mai complicat pentru dezvoltarea bazată pe SPL, acest lucru datorându-se parțial faptului că identificarea configurației este mai greu de realizat. Identificarea configurației este activitatea prin care sunt specificate numele, atributele și relațiile dintre configurații. În cazul dezvoltării normale configurația are o structură simplă, dar în cazul SPL, fiecare componentă care face parte din nucleul SPL-ului, fiecare componentă specializată și fiecare produs are o configurație care trebuie identificată, în plus trebuie specificate și relațiile dintre aceste configurații.

O posibilă abordare pentru managementul configurației software pentru un SPL constă în folosirea unei linii de dezvoltare pentru fiecare componentă de bază, respectiv pentru fiecare produs. Fiecare versiune de produs conține componentele specializate, precum și versiunile componentelor de bază. Sistemul de control al versiunii asigură faptul că, componentele de bază sunt doar citite și că ele nu sunt modificate în contextul unui produs. Totuși linia de dezvoltare a unui produs poate să folosească mai târziu o versiune mai nouă a componentei, care însă a fost dezvoltată în linia de dezvoltare a componentei.

9.3.3 Mecanisme Care Asigură Varietatea

Într-un SPL, componentele de bază oferă suport pentru funcționalitate variabilă prin intermediul punctelor de variație. Într-un SPL suportul pentru varietate este specificat chiar de la nivelul arhitecturii. Totuși, pot fi folosite și mecanisme de variație care sunt non-arhitecturale. În afară de mecanismele de variație de la nivelul arhitecturii există și mecanisme de variație la nivelul proiectării și la nivelul codului sursă. Aceste trei tipuri de variații nu sunt incompatibile, astfel la un moment dat se pot folosi atât mecanisme de variație la nivelul fișierelor cât și la nivelul arhitecturii.

Variație la nivelul arhitecturii. Mecanismele de variație la nivelul arhitecturii sunt strategii de proiectare la nivel superior care au scopul de a permite sistemului software să suporte un anumit set de funcționalități. Aceste strategii sunt slab legate de facilitățile oferite de un anumit limbaj de programare. De exemplu arhitecturile de platforme software și plug-in-uri.

Variație la nivelul proiectului. Granița dintre arhitectură și proiect nu este întotdeauna bine delimitată. Mecanisme de variație la nivelul proiectului sunt cele care sunt suportate direct prin facilitățile puse la dispoziție de limbajul de programare folosit. Mecanismele de variație la nivelul arhitecturii trebuie create prin programare, ele nu sunt suportate direct de limbajul de programare folosit. Exemple de mecanisme de variație la nivelul proiectului sunt: definirea de interfețe pentru componentele software care pot fi implementate în diferite moduri, suportul pentru moștenire și supra-încarcare permite obiectelor să conțină funcționalitate diferită care satisface clasele de bază.

Variație la nivelul fișierelor. Mediile de dezvoltare și limbajele de programare oferă mecanisme prin care se poate implementa variația la nivelul codului sursă. Astfel unele limbaje de programare oferă suport pentru compilare condiționată și macro definiții. De asemenea scripturile de compilare pot realiza variație logică sau fizică la nivelul fișierelor, acestea pot fi folosite pentru a varia funcționalitatea.

Variație la nivelul managementului configurației software. Principalul rol al managementului configurației software într-un SPL este acela de a oferi suport pentru refolosirea componentelor software prin identificarea și gestionarea versiunii unui produs și a componentelor care formează respectivul produs. Noi versiuni de produse nu trebuie obligatoriu să folosească cea mai nouă versiune a unei componente de bază. Managementul configurației software permite ca un produs să folosească orice versiune a unei componente de bază. Istoricul versiunilor și ramificarea versiunilor dintr-un sistem care gestionează configurația software, pot fi folosite pentru a reprezenta variația.

9.3.4 Evoluția Arhitecturii Pentru un SPL

Atunci când se folosește dezvoltarea bazată pe SPL, evoluează atât componentele specifice unui anumit produs cât și componentele de bază ale SPL-ului. PLA reprezintă baza arhitecturală care asigură variația la nivelul componentelor de bază. Astfel, modificarea interfețelor de bază reprezintă de fapt o modificare a PLA-ului și poate să necesite modificări în toate produsele care folosesc noua versiune a componentelor de bază. Se pune astfel întrebarea când ar trebui adăugate funcționalități noi sau îmbunătățiri într-un SPL.

Există trei momente de timp la care noi funcționalități pot fi introduse:

- **Proactiv** – planificarea în avans pentru funcționalități care vor fi folosite ulterior, respectiv implementarea acestor înainte ca un produs să aibă nevoie de ele;
- **Reactiv** – implementarea unei funcționalități noi se face în momentul în care un produs are nevoie de ea; funcționalitatea este implementată direct în nucleul SPL-ului;
- **Retroactiv** – implementarea unei funcționalități în momentul în care un produs are nevoie de ea; implementarea se face în contextul produsului care are nevoie de respectiva funcționalitate, atunci când un număr suficient de mare implementează aceeași funcționalitate ea este adăugată în nucleul SPL-ului, astfel că ea va putea fi folosită de alte produse care nu au implementat-o deja, iar produsele pentru care a fost implementată pot să renunțe la vechea componentă.

În realitate de cele mai multe ori se recurge la un mix între cele trei tipuri de strategii de adăugare de noi funcționalități la un SPL. Fiecare din cele trei strategii prezintă diferite costuri, riscuri și beneficii. În Tabela 9.1 este prezentat un rezumat al diferențelor dintre cele trei startegii.

	Proactiv	**Reactiv**	**Retroactiv**
Fără investiții pe termen lung	nu	da	da
Reduce riscul de conflicte între modificările necesare pentru componentele de bază	da	nu	da
Reduce timpul necesar adăugării une noi funcționalități pentru primul produs care o cere	da	nu	nu
Reduce riscul ca o anumită funcționalitate să nu fie folosită de mai multe produse	nu (0 produse)	nu (1 produs)	da

Tabela 1. Compararea strategiilor de evoluție a arhitecturii unui SPL.

Bibliografie

[1] Ian Gorton. Essential Software Architecture. Editura Springer. 2006.