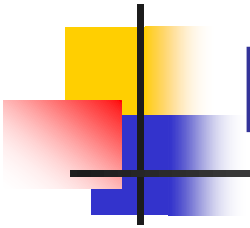
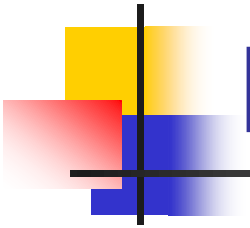
Paradigme de dezvoltare software



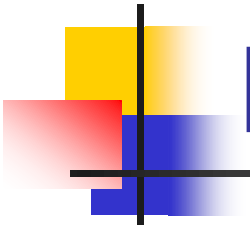
Paradigme

- In acest capitol sunt prezentate paradigme software de baza si evolutive.
- Prezentarea este adaptata (in special) dupa [Pre97,Pre00,Som89,Som01,Som04,Som06,Som10].
- O paradigma software furnizeaza o reprezentare abstracta a unui model de proces software.



Paradigme

- Notiunea de **process software** denota activitatile necesare obtinerii unui produs software care satisface cerinte date.
- O **paradigma de dezvoltare software** este un model foarte general de proces software, reprezentat de obicei din perspectiva arhitecturala.



Paradigme

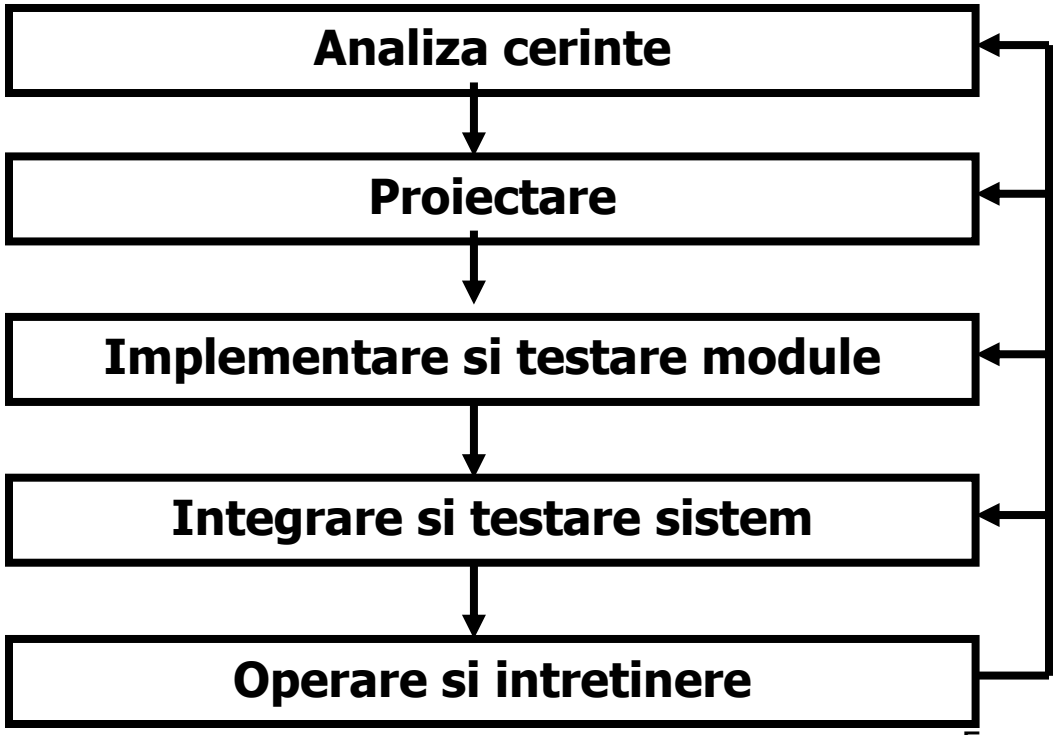
- Sunt prezentate patru **pardigme de baza**:
  - modelul linear (‘cascada’),
  - prototipizare,
  - componente reutilizabile si
  - metode formale.
- Sunt prezenate urmatoarele paradigme evolutive care furnizeaza suport pentru **procese iterative**:
  - modelul incremental,
  - modelul spirala si
  - modelul ingineriei concurente.

Paradigme de baza – modelul linear

Modelul linear (numit uneori 'cascada') [Roy70]

- Dezvoltarea software in acest model cuprinde urmatoarele stadii:

- Acest model este util in proiectele in care cerintele sunt bine intelese de la inceput.
- In acest model insa, este dificil sa se reactioneze la modificarea cerintelor (la solicitarea clientului) deoarece toate deciziile de analiza si design sunt luate la inceputul proiectului.
- Modelul linear este utilizat in multe proiecte mici si medii. In proiecte mari este de obicei utilizat ca parte a unui proces software iterativ.

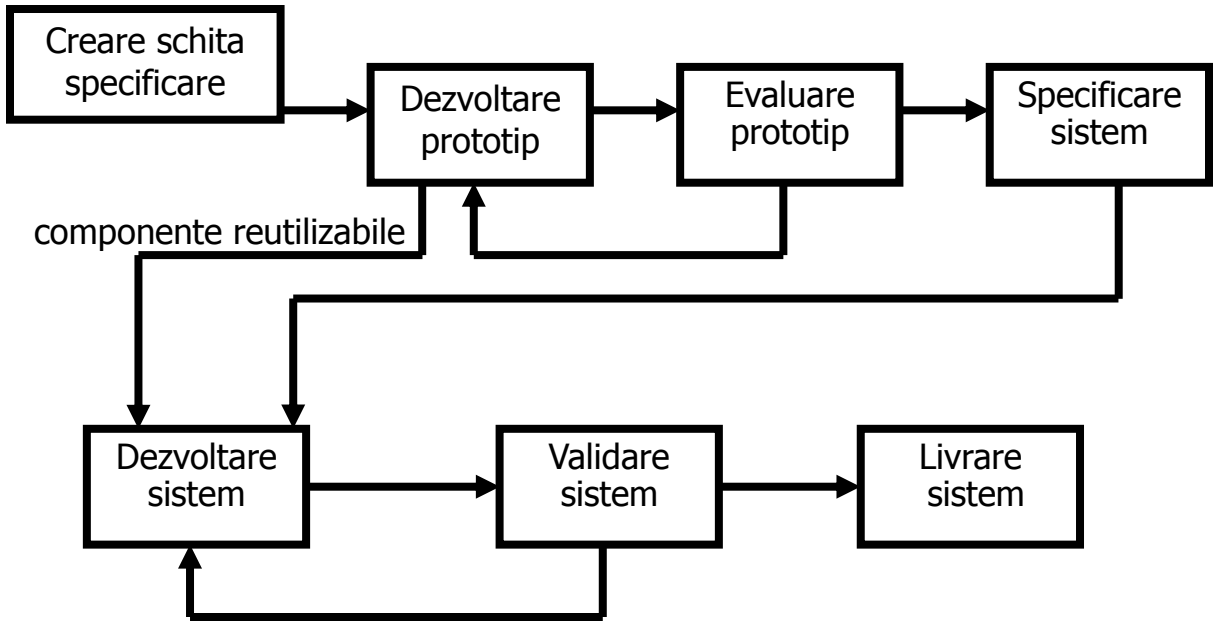


Paradigme de baza – prototipizare

Dezvoltare prin prototipizare

- Un **prototip** este o implementare initiala a unui sistem, care serveste ca mecanism de identificare a cerintelor software.

- Dupa stabilirea cerintelor se poate renunta la prototip (o parte dintre componentele prototipului se pot reutiliza in sistemul final).
- Urmeaza o reimplementare a sistemului software cu scopul de a se realiza un produs ce satisace criterii de performanta date.



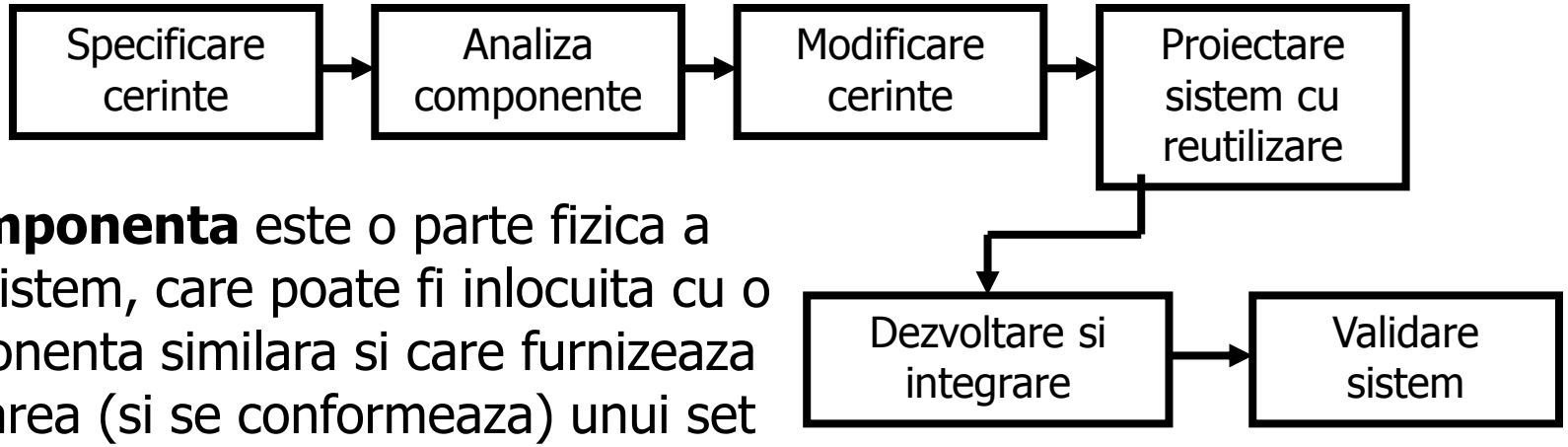
Paradigme de baza – prototipizare

- Prototipul are numai rolul de a demonstra aspectele functionale ale sistemului (nu si pe cele nonfunctionale).
- Este necesar ca prototipul sa poata fi dezvoltat la un cost semnificativ mai mic decat costul sistemului final. Pentru reducerea costurilor aferente construirii prototipului [Som01,Som06,Som89]:
  - se poate renunta la o parte din functionalitate si se pot relaxa cerintele nonfunctionale;
  - in realizarea prototipului se pot utiliza limbaje de programare de nivel foarte inalt (Lisp, Prolog si Smaltalk sunt exemple clasice);
  - se ignora tratarea conditiilor de eroare / exceptie.
  - se pot folosi medii vizuale si generatoare automate de cod;
  - se pot reutiliza componente existente.

Paradigme de baza – componente reutilizabile

Modelul de **dezvoltare orientata pe componente** are drept caracteristica principala **reutilizabilitatea**.

- Aceasta abordare presupune existenta unor biblioteci de componente si a unui cadru de dezvoltare, reutilizare si integrare.



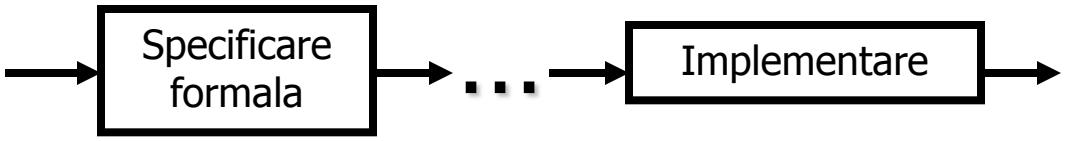
O **componenta** este o parte fizica a unui sistem, care poate fi inlocuita cu o componenta similara si care furnizeaza realizarea (si se conformeaza) unui set de interfete [JBR99].

Paradigme de baza – componente reutilizabile

- Stadiile procesului de dezvoltare bazata pe componente reutilizabile sunt urmatoarele [Som06]:
  - Data fiind o **specificare a cerintelor**, in faza de **analiza componente** se cauta componente care sa implementeze respectiva specificare. Adesea, nu exista o potrivire exacta iar componentele disponibile furnizeaza numai o parte din functionalitatea ceruta.
  - **Modificare cerinte** este o faza in care cerintele sunt analizate si, daca este necesar, sunt modificate pentru a reflecta functionalitatea componentelor disponibile. Acolo unde cerintele nu se pot modifica, se reia faza de analiza componente in scopul de obtinere a unor solutii alternative.
  - In faza de **proiectare sistem prin reutilizare**, se proiecteaza sistemul tinand cont de componentele reutilizabile.
  - In faza de **dezvoltare si integrare**, se dezvolta modulele software care nu pot fi cumparate si componentele sunt integrate pentru a se crea sistemul.

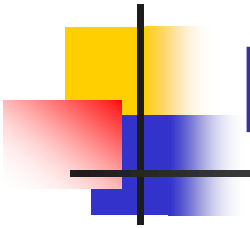
Paradigme de baza – metode formale

- In **modelul de dezvoltare prin metode formale** sistemul software este construit pe baza unei specificari matematice (formale).
  - O **specificare formala** furnizeaza o descriere precisa si independenta de implementare a comportamentului sistemului, putand fi privita drept (un prototip matematic sau) o 'schita matematica' a sistemului.
  - Trecerea de la specificarea formala la implementare se poate realiza prin transformari formale (care pastreaza semantica matematica, dar un asemenea proces poate fi costisitor) sau se poate realiza in mod informal.
  - Pentru sisteme mari, pasul de specificare formala este precedat de o faza de proiectare arhiecturala, iar pasul de implementare este urmat de o faza de testare si integrare la nivel de sistem.



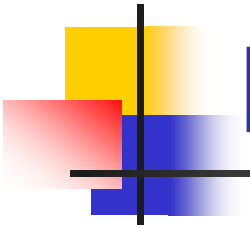
Paradigme de baza – metode formale

- Specificarea matematica face posibila **verificarea formala** a corectitudinii software, ceea ce conduce la o reducere semnificativa a costurilor aferente testarii. In principiu, prin verificare formala se poate elimina testarea la nivel de modul, aspect important deoarece
- Testarea este mare consumatoare de resurse:
  - 30%-40% in proiecte obisnuite;
  - pana la de 5 ori resursele necesare tuturor celorlalte activitati de dezvoltare in proiecte care necesita fiabilitate foarte ridicata (cf. [Pre97]).
- Metodele formale sunt potrivite pentru proiecte cu cerinte stringente in ceea ce priveste **siguranta**, **securitatea** sau **fiabilitatea**, cum sunt sistemele de control trafic aerian, sau sistemele medicale.
- Exemple de metode si abordari formale:
  - Specificare bazata pe model in notatia Z [Crr99,Jac97,Spi92]
  - Model-checking [BK08]
  - Specificare algebrica [EM85, EM90]



Paradigme evolutive

- Sistemele software de dimensiuni mari evolueaza in timp si cerintele se modifica adesea pe parcursul procesului de dezvoltare.
- Inginerii software au nevoie de modele de procese care sa ofere un cadru de dezvoltare pentru sistemele care evolueaza in timp.
- Dar paradigmele de baza nu iau in considerare natura evolutiva a sistemelor software.



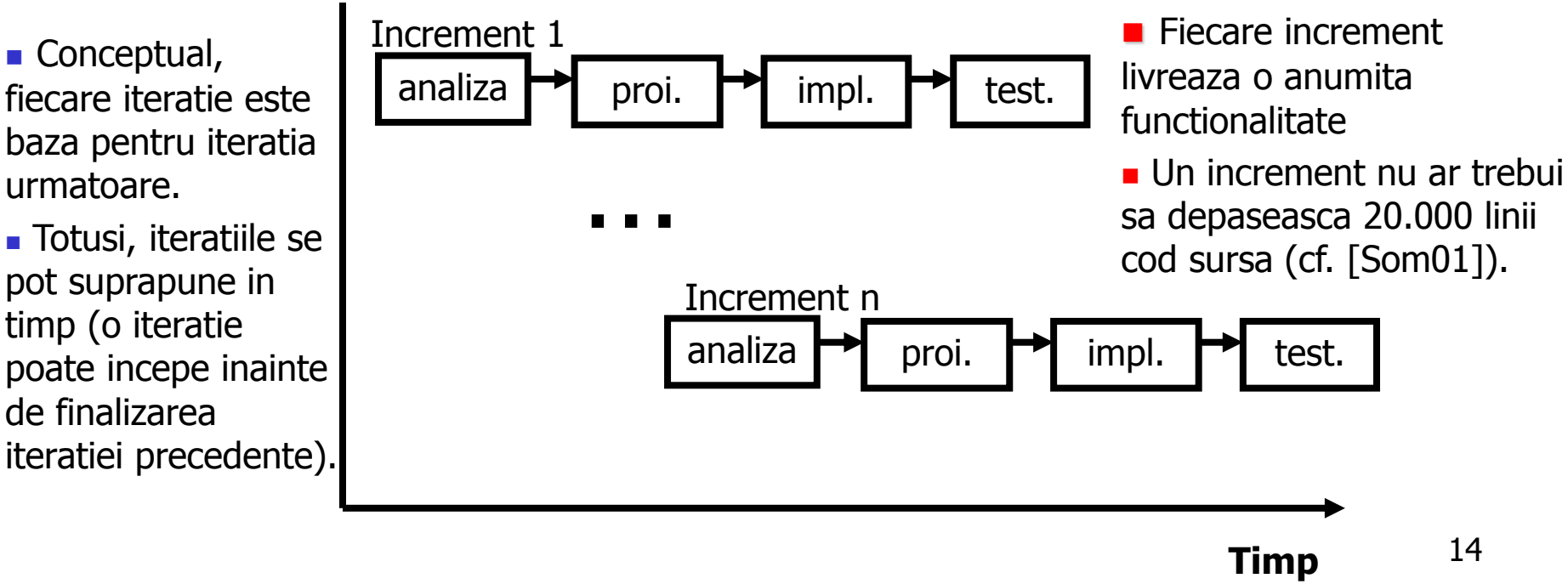
Paradigme evolutive

- In cele ce urmeaza prezentam trei **modele evolutive**:
  - modelul incremental,
  - modelul spirala si
  - modelul ingineriei concurente.
- Esenta modelelor evolutive:
  - specificarea se dezvolta in conjunctie cu sistemul software [Som01,Som06,Som10]
  - paradigmele evolutive sunt iterative.

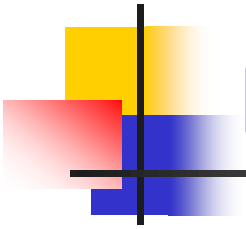
Paradigme evolutive – modelul incremental

Modelul incremental

- Utilizat in RUP [JBR99,Kru03] si XP [Bec99]
- Imparte proiectul in parti de dimensiuni controlabile, numite *incremente* (figura este din [Pre97]):



Paradigme evolutive – modelul incremental

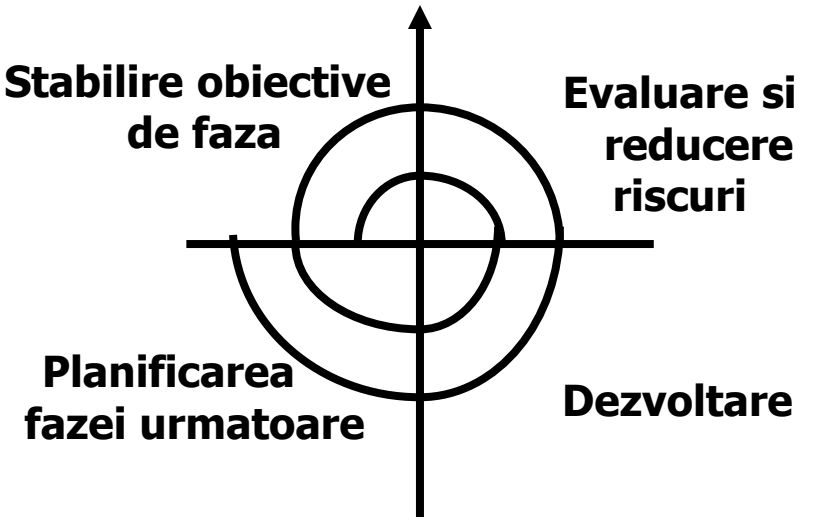


- In figura s-a presupus ca in fiecare increment se utilizeaza modelul linear (cascada). In general insa, fiecare increment poate incorpora orice paradigma potrivita [Som01].
- Avantaje ale dezvoltarii incrementale [Som01,JBR99]:
  - clientii nu trebuie sa astepte pana cand intregul sistem este livrat; primele incremente satisfac cele mai critice cerinte, deci produsul software poate fi deja utilizat;
  - riscurile sunt reduse la costul unui singur increment.
- Principala dificultate in dezvoltarea incrementala [Som01,Som06,Som10]:
  - poate fi dificil sa se mapeze cerintele clientului la incremente de dimensiuni controlabile.

Paradigme evolutive – modelul spirala

In **modelul spirala** [Boe88] (vezi si [Pre97,Som06]) produsul software este dezvoltat intr-o serie de versiuni incrementale, iar procesul este reprezentat ca o spirala evolutiva. **Riscurile** sunt considerate si tratate in fiecare faza (bucla) a spiralei.

- Fiecare faza (bucla) a spiralei este impartita in zone:

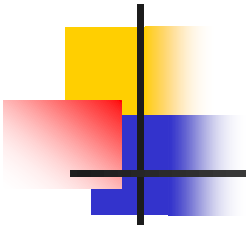


Prototipizarea poate fi utilizata ca mecanism de reducere a riscurilor.

Riscurile identificate permit alegerea celei mai potrivite pardigme de dezvoltare pentru fiecare faza.

Paradigme evolutive – inginerie concurenta

- Modelul de **dezvoltare concurenta**, numit uneori **inginerie concurenta** [DP94,Pre97,She94], furnizeaza o descriere precisa a starii curente a unui proiect; proiectul este reprezentat ca o retea de activitati concurente si comunicante.
  - Un eveniment (de exemplu o modificare tarzie a cerintelor) din cadrul unei activitati (de exemplu **analiza**) poate cauza o tranzitie de la o stare la alta a unei alte activitati (de exemplu activitate de **design** trece din starea de dezvoltare intr-o starea de asteptare schimbari).
- De obicei, pentru reprezentarea activitatilor concurente in acest model se utilizeaza **diagrame de tranzitie de stare**. Acest model descrie procesul software ca o colectie de automate concurente si comunicante.



Bibliografie

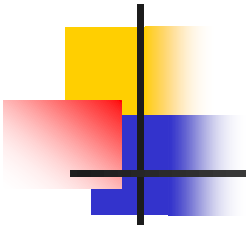
[BK08] C. Baier, J.-K. Katoen. Principles of Model Checking. The MIT Press Cambridge, 2008.

[Bec99] K. Beck. Embracing Change with Extreme Programming. *IEEE Computer* 32(10):70-78, 1999.

[Boe88] B. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61-72, 1988.

[Crr99] E. Currie. *The Essence of Z*. Prentice Hall, 1999.

[DP94] A. Davis, P. Pitaram. A concurrent process model for software development. *Software Engineering Notes*, ACM Press, 19(2):38-51, 1994.



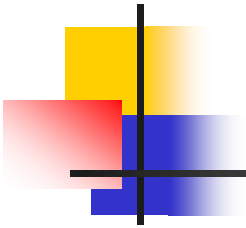
Bibliografie

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification, part 1*. Springer, 1985.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification, part 2*. Springer, 1990.

[Jac97] J. Jacky. *The Way of Z: Practical Programming with Formal Methods*. Cambridge University Press, 1997.

[JBR99] I. Jacobson, G. Booch and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.



Bibliografie

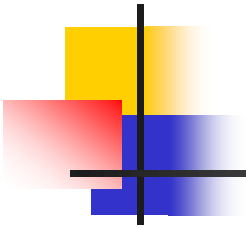
[Kru03] P. Kruchten. *The Rational Unified Process: An Introduction* (3rd edition). Addison-Wesley, 2003.

[Pre97] R. Pressman. *Software Engineering: A Practitioner's Approach*, (4th edition). McGraw-Hill, 1997.

[Pre00] R. Pressman. *Software Engineering: A Practitioner's Approach*, (5th edition). McGraw-Hill, 2000.

[Pre14] R. Pressman. *Software Engineering: A Practitioner's Approach*, (8th edition). McGraw-Hill, 2014.

[Roy70] W. Royce. Managing the development of large software systems: concepts and techniques. In *Proc. IEEE WESTCON*, 1970.



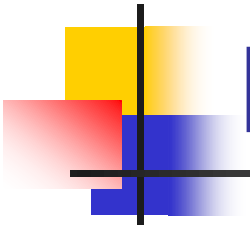
Bibliografie

[She94] W. Sheleg. Concurrent Engineering: A New Paradigm for C/S Development. *Application Development Trends*, 1(6):28-33, 1994.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, 1992.

[Som89] I. Sommerville. *Software Engineering,* (3rd edition). Addison-Wesley, 1989.

[Som01] I. Sommerville. *Software Engineering,* (6th edition). Addison-Wesley, 2001.



Bibliografie

[Som04] I. Sommerville. *Software Engineering,* (7th edition). Addison-Wesley, 2004.

[Som06] I. Sommerville. *Software Engineering,* (8th edition). Addison-Wesley, 2006.

[Som10] I. Sommerville. *Software Engineering,* (9th edition). Addison-Wesley, 2010.

[Som15] I. Sommerville. *Software Engineering,* (10th edition). Addison-Wesley, 2015.