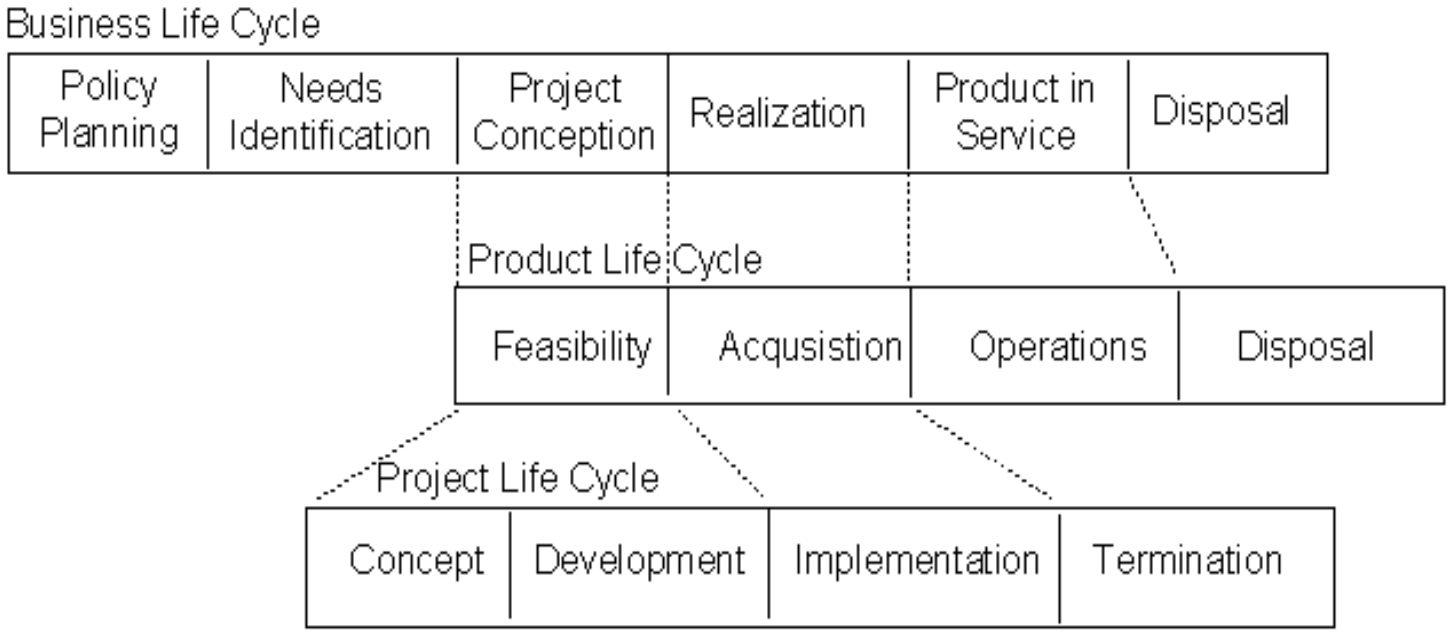
Università
Ca' Foscari
Venezia

MODELLI DI PROCESSO DI PRODUZIONE SOFTWARE

Università Ca' Foscari Venezia

Cicli di vita: business, prodotto e processo



Modelli di processo

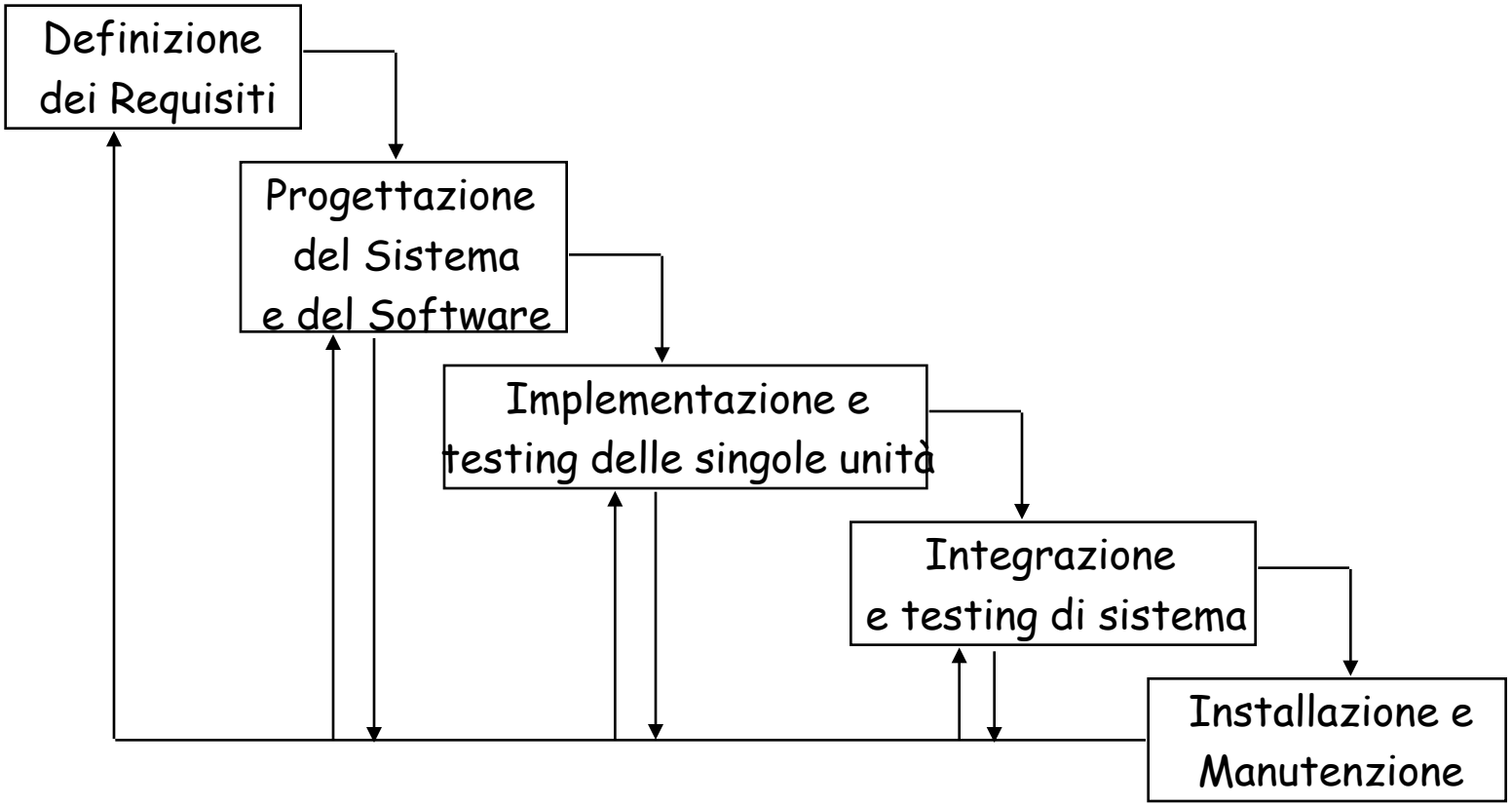
1. Modello a cascata
   - Fasi distinte di specifica e di sviluppo
2. Modello evolutivo
   - Specifica e sviluppo interagiscono
3. Modello trasformazionale
   - Un sistema matematico è trasformato formalmente in una implementazione

Modelli di processo

4. Sviluppo basato sul riutilizzo
   - Il sistema è ottenuto combinando componenti esistenti
5. Sviluppo Agile (eXtreme Programming)
   - Sviluppo e rilascio di incrementi molto piccoli di funzionalità
6. Modello a spirale
   - Si parte dai rischi

Università
Ca' Foscari
Venezia

1. Modello a cascata



Fasi del modello a cascata

- Analisi e definizione dei requisiti
- Progettazione del sistema e del software
- Implementazione e test delle singole unità
- Integrazione e test del sistema
- Installazione e mantenimento

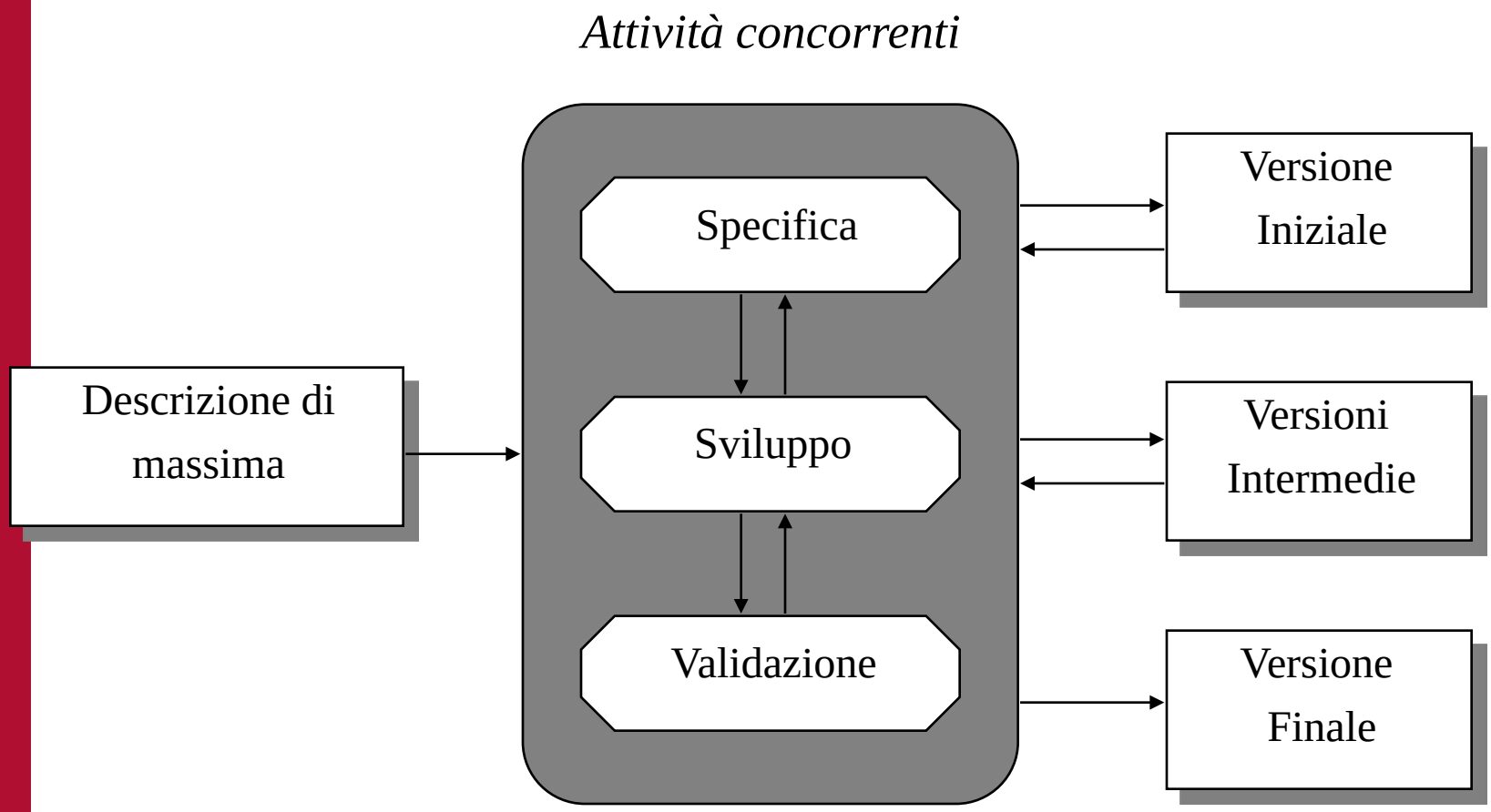
- Il limite del modello a cascata è la difficoltà ad effettuare cambiamenti nel corso del processo

Documentazione del modello a cascata

Activity	**Output documents**
Requirements analysis	Feasibility study, Outline requirements
Requirements definition	Requirements document
System specification	Functional specification, Acceptance test plan Draft user manual
Architectural design	Architectural specification, System test plan
Interface design	Interface specification, Integration test plan
Detailed design	Design specification, Unit test plan
Coding	Program code
Unit testing	Unit test report
Module testing	Module test report
Integration testing	Integration test report, Final user manual
System testing	System test report
Acceptance testing	Final system plus documentation

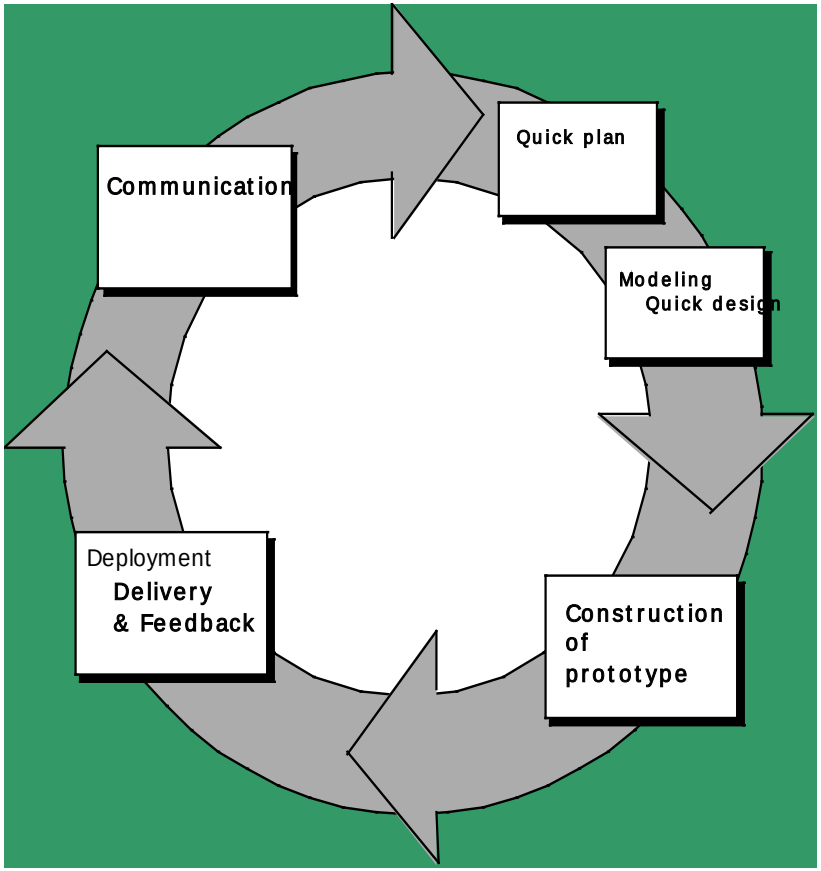
Università Ca' Foscari Venezia

2. Modello evolutivo



Università
Ca' Foscari
Venezia

Modello evolutivo



Modello evolutivo

- Prototipazione di tipo evolutivo
  - L'obiettivo è lavorare con il cliente ed evolvere verso il sistema finale a partire da una specifica di massima. Lo sviluppo inizia con le parti del sistema che sono già ben specificate, aggiungendo via via nuove caratteristiche
- Prototipazione di tipo usa e getta
  - L'obiettivo è capire i requisiti del sistema. e quindi sviluppare una definizione migliore dei requisiti. Il prototipo sperimenta le parti del sistema che non sono ancora ben comprese

Modello evolutivo

- Problemi
  - Mancanza di visibilità del processo
  - Sistemi spesso poco strutturati
  - Possono essere richieste particolari capacità (ad esempio in linguaggi per prototyping rapido)
- Applicabilità
  - Sistemi interattivi di piccola o meda dimensione
  - Per parti di sistemi più grandi (es. interfaccia utente)
  - Per sistemi a vita breve

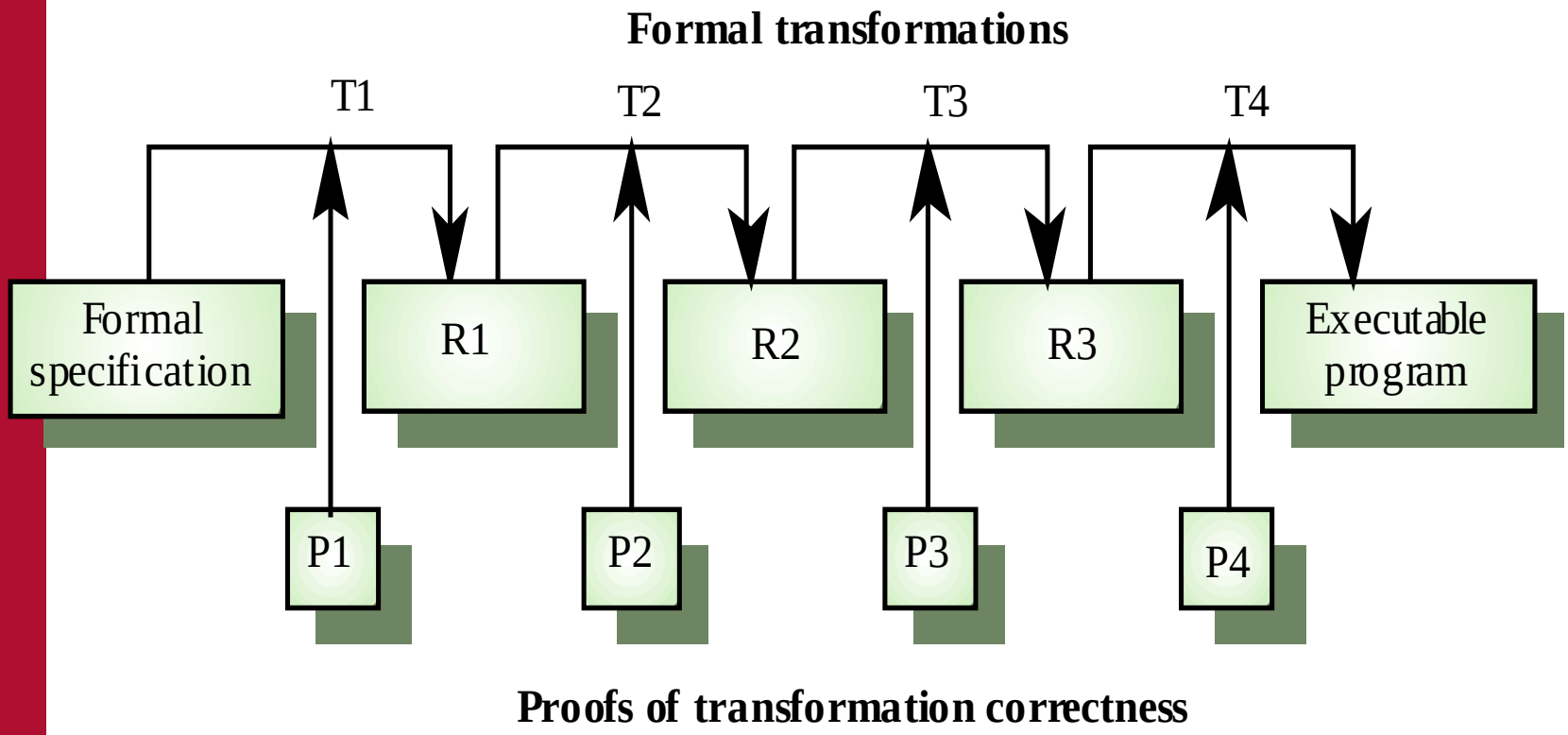
Università Ca' Foscari Venezia

3. Modello trasformazionale

- Basato sulla trasformazione di una specifica matematica in in programma eseguibile, attraverso trasformazioni che permettono di passare da una rappresentazione formale ad un'altra.
- Le trasformazioni devono preservare la correttezza. Questo garantisce che il programma soddisfi la specifica.
- Utilizzato per componenti critiche dei sistemi

Università Ca' Foscari Venezia

Modello trasformazionale

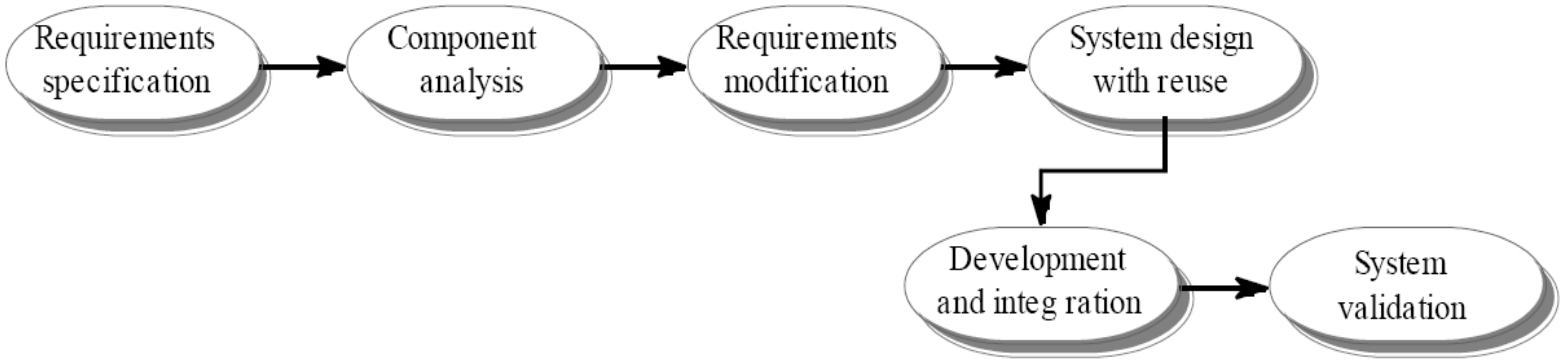


4. Modello basato sul riutilizzo

- Basato sul riuso sistematico di componenti off-the-shelf, integrate opportunamente
- Fasi del modello:
  - Analisi delle componenti
  - Adattamento dei requisiti
  - Progettazone del sistema
  - Integrazione

Università
Ca' Foscari
Venezia

Sviluppo basato sul riuso



5. Metodologie di Sviluppo Agile

- I principi su cui si basa una metodologia leggera che segua i punti indicati dall'Agile Manifesto, sono solo quattro:
  - **le persone e le interazioni sono più importanti dei processi e degli strumenti** (ossia le relazioni e la comunicazione tra gli attori di un progetto software sono la miglior risorsa del progetto)
  - è più importante avere **software funzionante** che documentazione (bisogna rilasciare nuove versioni del software ad intervalli frequenti, e bisogna mantenere il codice semplice e avanzato tecnicamente, riducendo la documentazione al minimo indispensabile)
  - bisogna **collaborare** con i clienti al di là del contratto (la collaborazione diretta offre risultati migliori dei rapporti contrattuali)
  - bisogna essere pronti a **rispondere ai cambiamenti** più che aderire al progetto (quindi il team di sviluppo dovrebbe essere autorizzato a suggerire modifiche al progetto in ogni momento)

XP Core Practice

- XP è definito dalle pratiche usate.
- Le pratiche variano nel tempo e a seconda del progetto in cui vengono utilizzate:

1. Planning the game
2. Simple Design
3. Pair Programming
4. Testing
5. Refactor
6. Short releases
7. Coding Standard

XP Core Practice: planning the game

- sviluppo dell'applicazione accompagnato dalla stesura di un piano di lavoro
- piano definito e aggiornato a intervalli brevi e regolari dai responsabili del progetto, secondo le priorità aziendali e le stime dei programmatori
- i programmatori partecipano, in modo attivo, alla pianificazione
- la pianificazione coinvolge sia utenti responsabili del progetto che sviluppatori per stabilire un equilibrio dinamico fra le esigenze di tutti

XP Core Practice: planning the game

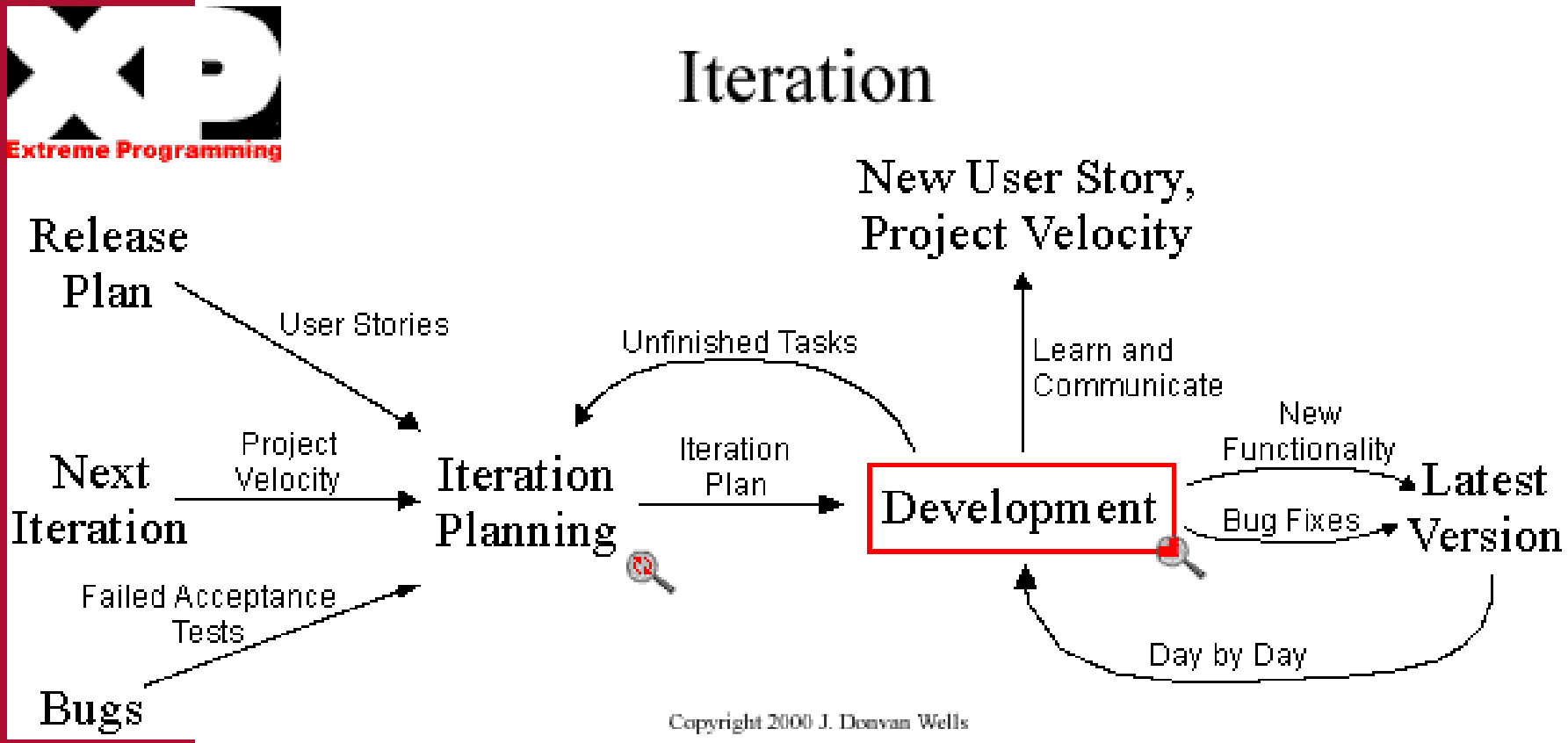
cont.

- gli utenti finali dell'applicazione presentano gli obiettivi da raggiungere descrivendo una serie di scenari (storie)
- gli sviluppatori stimano il tempo necessario per la realizzazione di ogni storia
- le storie vengono ordinate da utenti e responsabili secondo la loro priorità di realizzazione, dopo che gli sviluppatori ne hanno stimata la rispettiva difficoltà
- dalla sintesi delle valutazioni i responsabili del progetto generano la pianificazione delle attività, intesa come l'insieme di storie che dovranno essere realizzate per il prossimo rilascio e le date previste

Università
Ca' Foscari
Venezia

XP Core Practice: planning the game

cont.



XP Core Practice: simple design

- la struttura dell'applicazione deve essere la più semplice possibile
- l'architettura del sistema deve essere comprensibile da tutte le persone coinvolte nel progetto
- non devono esserci parti superflue o duplicazioni
- le parti che compongono il sistema devono essere soltanto quelle strettamente necessarie alle esigenze correnti
- solo quando nuove circostanze lo richiederanno, verranno progettati nuovi componenti, eventualmente riprogettando anche quelli già esistenti

Università Ca' Foscari Venezia

XP Core Practice: pair programming

- la scrittura vera e propria del codice è fatta da coppie di programmatori che lavorano al medesimo terminale
- le coppie non sono fisse, ma si compongono associando migliori competenze per la risoluzione di uno specifico problema
- il lavoro in coppia permette, scambiandosi periodicamente i ruoli, di mantenere mediamente più alto il livello d'attenzione
- i locali dove si svolge il lavoro devono permettere senza difficoltà di lavorare a coppie

XP Core Practice: testing

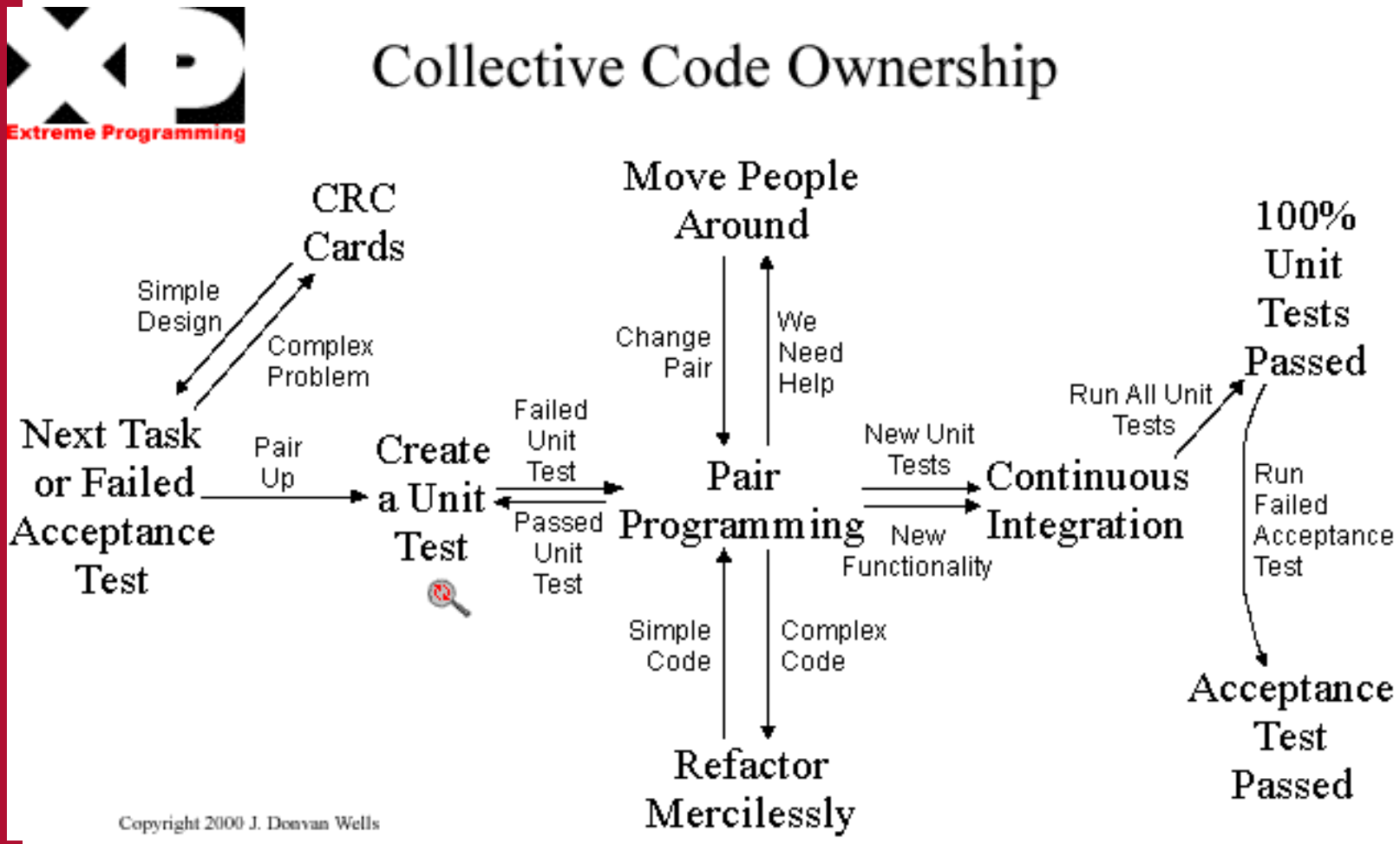
- ogni funzionalità va sottoposta a verifica, in modo che si possa acquisire una ragionevole certezza sulla sua correttezza
- test di sistema costruiti sulla base delle storie concordate con il committente
- test di unità che devono poter essere rieseguiti automaticamente, con tempi dell'ordine dei minuti
- ogni ristrutturazione o modifica del codice deve mantenere inalterato il risultato dei test già considerati
- i test vengono, generalmente, scritti prima della codifica della funzionalità

XP Core Practice: refactor

- soprattutto dopo molti cambiamenti nel tempo il codice diventa poco maneggevole
- i programmatori spesso continuano a utilizzare codice non più mantenibile perché continua a funzionare
- quando stiamo rimuovendo ridondanza, eliminiamo funzionalità non utilizzate e rinnoviamo un design obsoleto stiamo rifattorizzando
- il refactoring mantiene il design semplice, evita complessità inutili, mantiene il codice pulito e conciso così che sia facilmente comprensibile, modificabile e estendibile

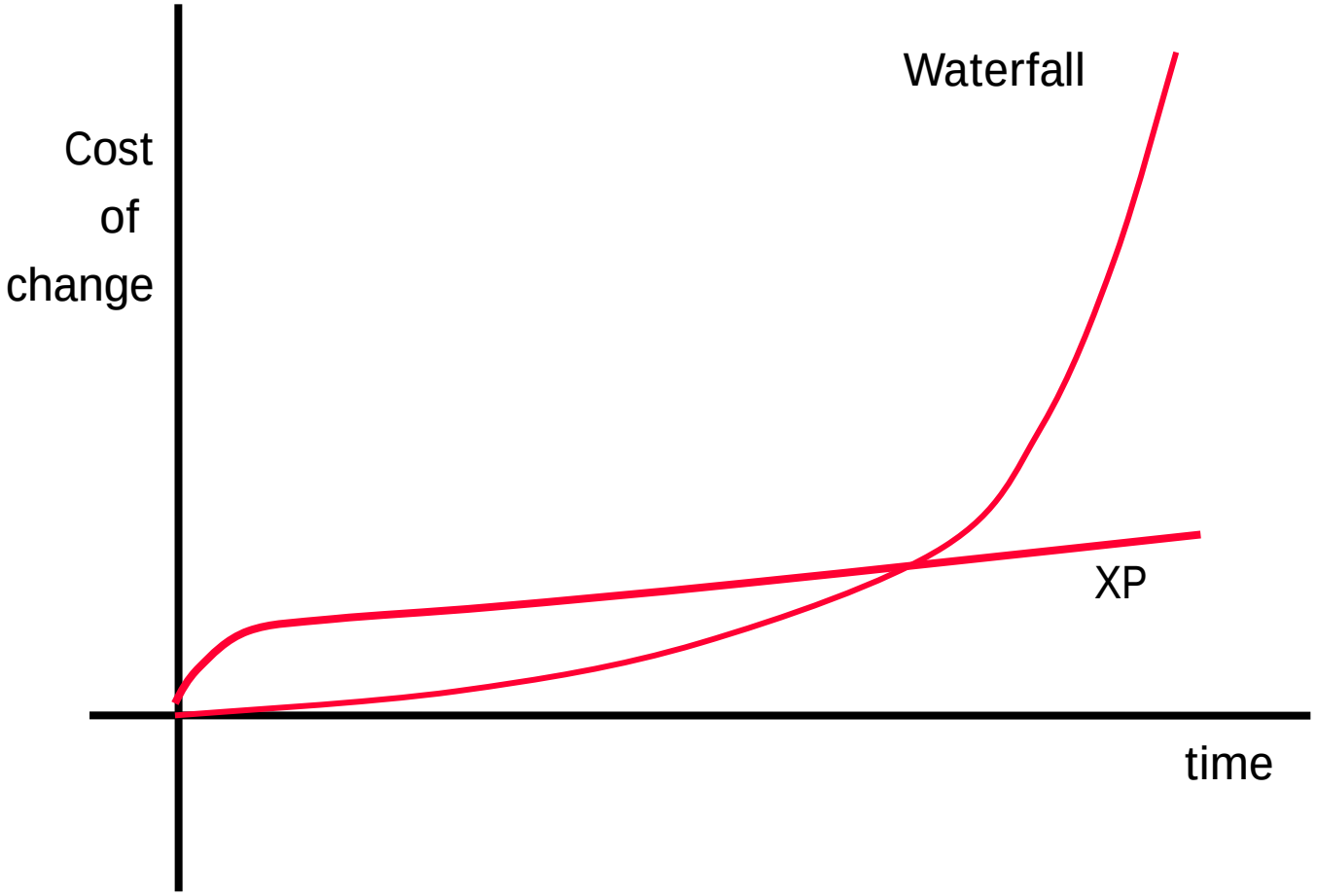
Università Ca' Foscari Venezia

XP Core Practice: collective code ownership



Università
Ca' Foscari
Venezia

Cost of Change



Università Ca' Foscari Venezia

6. Modello a spirale

- Obiettivo principale = minimizzare i rischi
- Rischio = misura di incertezza del risultato di un'attività
- Meno informazione si ha, più alti sono i rischi

Università
Ca' Foscari
Venezia

Modello a spirale di Boehm

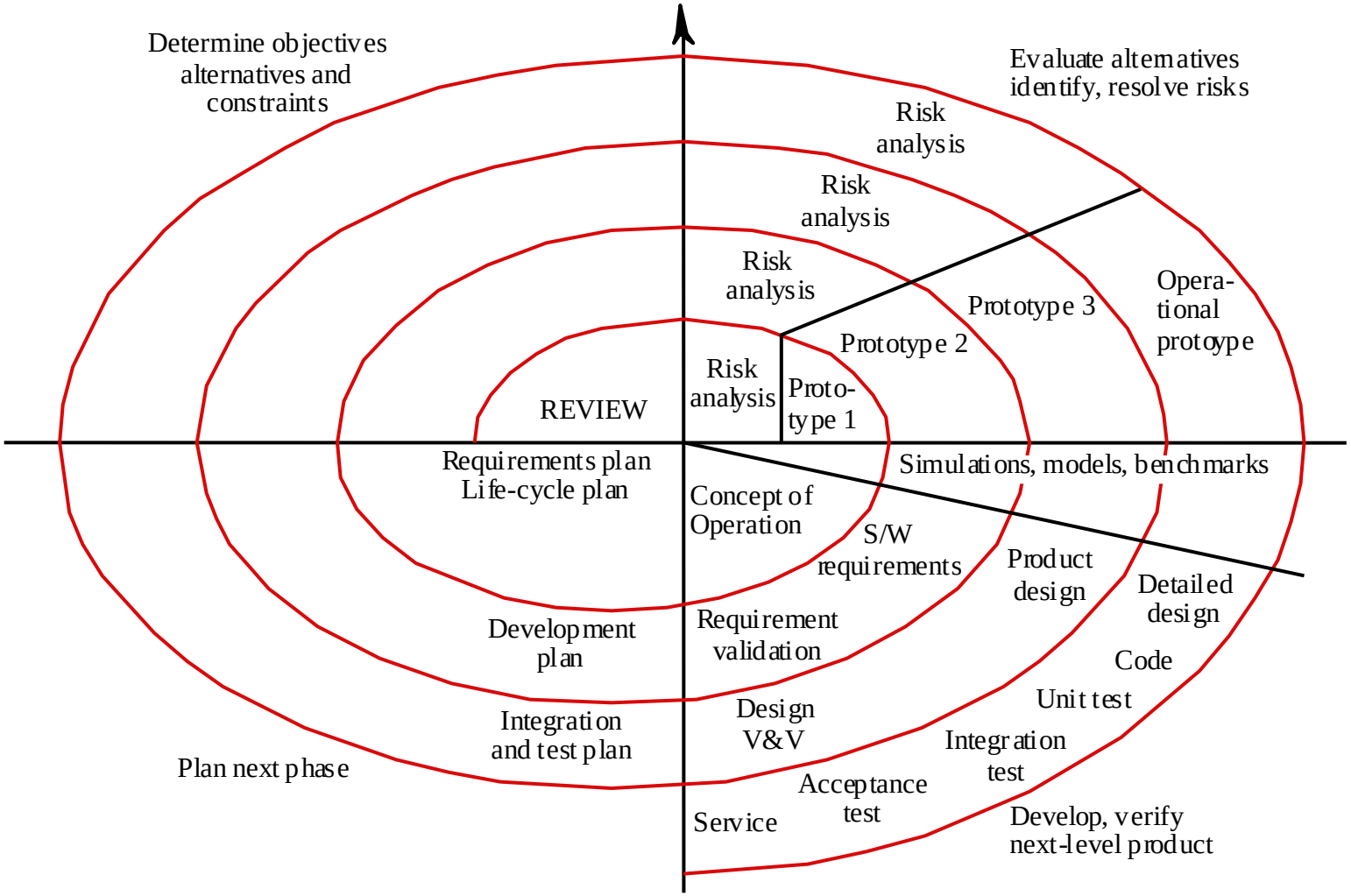
Università Ca' Foscari Venezia

Fasi del modello a spirale

- Determinazione degli obiettivi e dei vincoli
- Valutazione e riduzione dei rischi e valutazione delle alternative
- Progettazione e testing
- Pianificazione della fase successiva

Università Ca' Foscari Venezia

Modello a spirale



Vantaggi e limiti del modello a spirale

- Vantaggi
  - Concentra l'attenzione sulle possibilità di riuso
  - Concentra l'attenzione sull'eliminazione di errori
  - Pone al centro gli obiettivi
  - Integra sviluppo e mantenimento
  - Costituisce un framework di sviluppo hardware/software
- Limiti
  - Per contratto di solito si specifica a priori il modello di processo e i "deliverables"
  - Richiede esperienza nella valutazione dei rischi
  - Richiede raffinamenti per un uso generale

Valutazione dei rischi nei modelli di processo

- Modello a cascata
  - Alto rischio per sistemi nuovi, per problemi di specifica e di progettazione
  - Basso rischio per sviluppo di problemi familiarità già acquisita
- Modello evolutivo, prototipazione
  - Basso rischio per nuovi sistemi
  - Alto rischio a causa della scarsa visibilità del processo
- Modello trasformazionale
  - Alto rischio dovuto alla necessità di tecnologia avanzata e di elevate capacità da parte degli sviluppatori

Visibilità del processo software

- C'è bisogno di documentazione per valutare i progressi nel processo di sviluppo software
- Problemi
  - La programmazione dei tempi di consegna dei "deliverables" può non combaciare con i tempi necessari per completare un'attività
  - La necessità di produrre documentazione vincola l'iterazione del processo
  - Il tempo necessario per approvare i documenti è significativo
- Il modello a cascata è ancora il modello basato su "deliverables" più usato

Visibilità del processo nei vari modelli

Process model	Process visibility
Waterfall model	Good visibility, each activity produces some deliverable
Evolutionary development	Poor visibility, uneconomic to produce documents during rapid iteration
Formal transformations	Good visibility, documents must be produced from each phase for the process to continue
Reuse-oriented development	Moderate visibility, it may be artificial to produce documents describing reuse and reusable components.
Spiral model	Good visibility, each segment and each ring of the spiral should produce some document.