RPL101
Pengantar Rekayasa Perangkat Lunak
Introduction to Software Engineering

Semester Ganjil Tahun Ajaran 2024-2025

PROGRAM STUDI
TEKNOLOGI REKAYASA PERANGKAT LUNAK
POLIBATAM

7. Model Proses Perangkat Lunak
Software Process Model

PROGRAM STUDI
TEKNOLOGI REKAYASA
PERANGKAT LUNAK
POLIBATAM

Software Process

Merupakan sekumpulan aktivitas yang harus dilakukan untuk membangun sebuah perangkat lunak

Software process model (model proses perangkat lunak) merupakan penggambaran dari *software process*

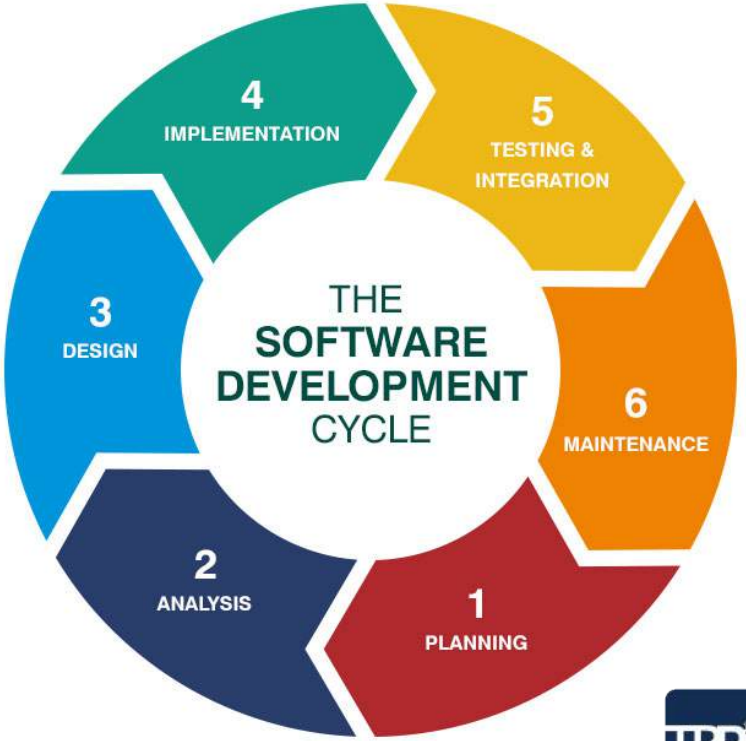
Plan-driven process adalah proses dimana semua aktivitas direncanakan di awal dan progress diukur berdasarkan rencana tersebut

Agile process merupakan proses dimana perencaan dilakukan secara bertahap sehingga lebih mudah mengubah proses tersebut sesuai kebutuhan customer

Tidak ada software process yang **benar** atau **salah**

Software Process Activities

- Planning
- Analysis
- Design
- Implementation
- Testing
- Maintenance

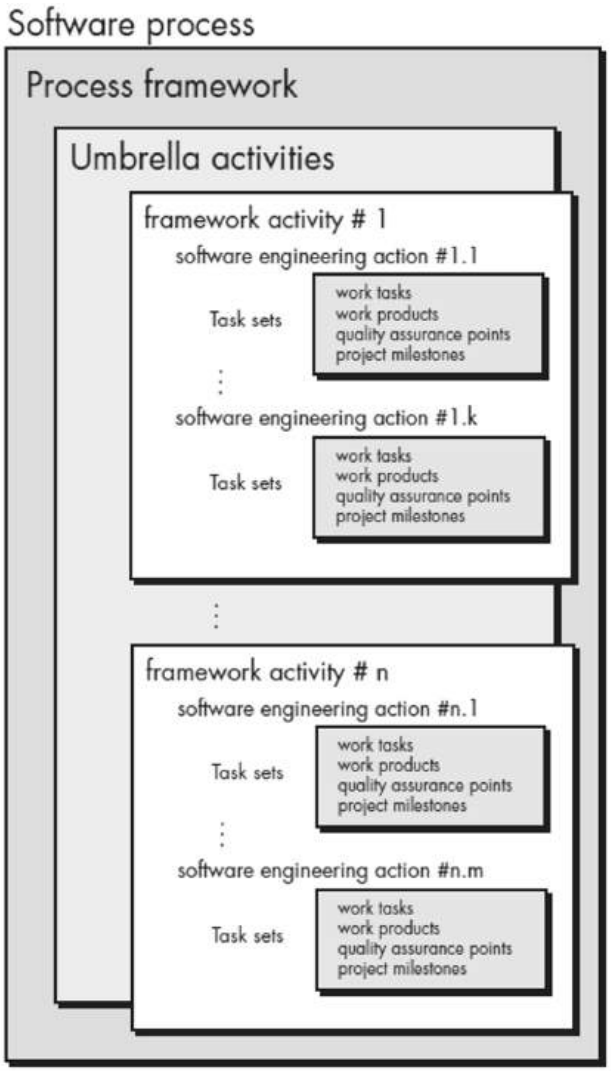


PROGRAM STUDI
TEKNOLOGI REKAYASA
PERANGKAT LUNAK
POLIBATAM

Model Proses

Secara umum, sebuah proses terdiri dari lima aktivitas—perencanaan, analisis, desain, implementasi, pengujian. Sebagai tambahan, beberapa aktivitas pendukung—project tracking and control, risk management, quality assurance, configuration management, technical reviews, dan sebagainya—juga dilakukan selama proses berlangsung.

Setiap aktivitas terdiri dari serangkaian aksi



Umbrella Activities (Pressman, 2009)

Software project tracking and control—untuk tim pengembang melacak progres/kemajuan proyek dan agar tim pengembang dapat mengambil tindakan yang diperlukan agar pengerjaan sesuai dengan jadwal.

Risk management—menentukan semua risiko yang mungkin dapat mempengaruhi kinerja proyek atau kualitas produk.

Software quality assurance—mendefinisikan dan menjalakan semua aktivitas untuk menjamin kualitas produk.

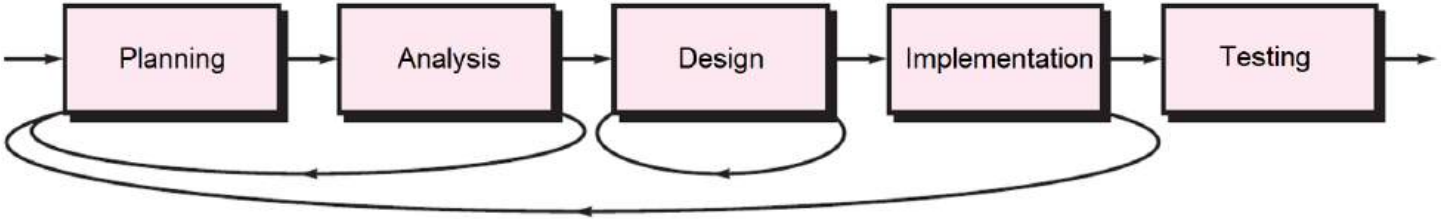
Software configuration management—mengelola efek atau akibat yang terjadi karena adanya perubahan selama berlangsungnya software process.

Reusability management—mendefinisikan kriteria untuk produk yang dapat digunakan kembali (termasuk software components) dan menentukan mekanisme untuk menghasilkan software component yang dapat digunakan kembali.

dsb…

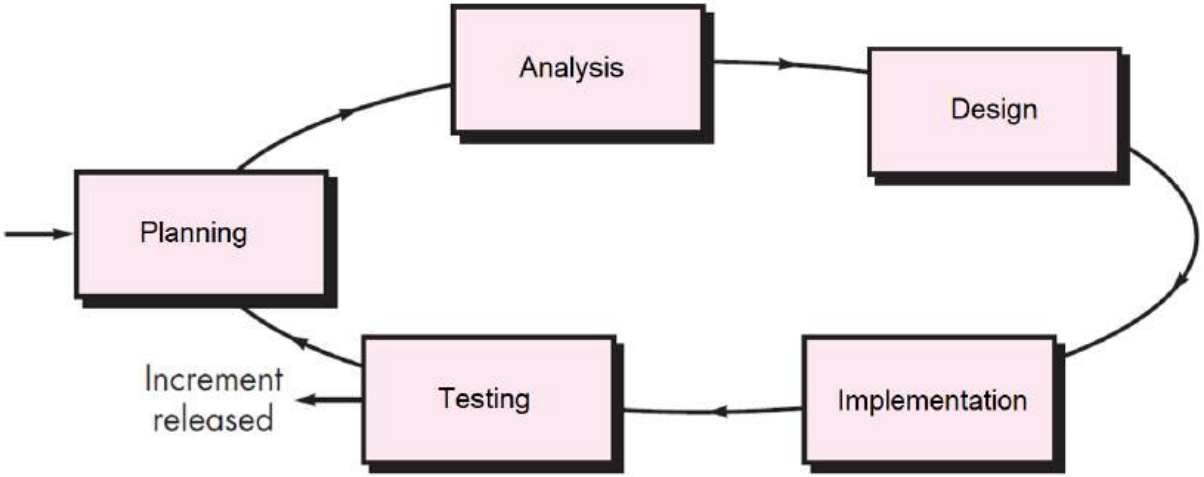
Process Flow

(a) Linear process flow



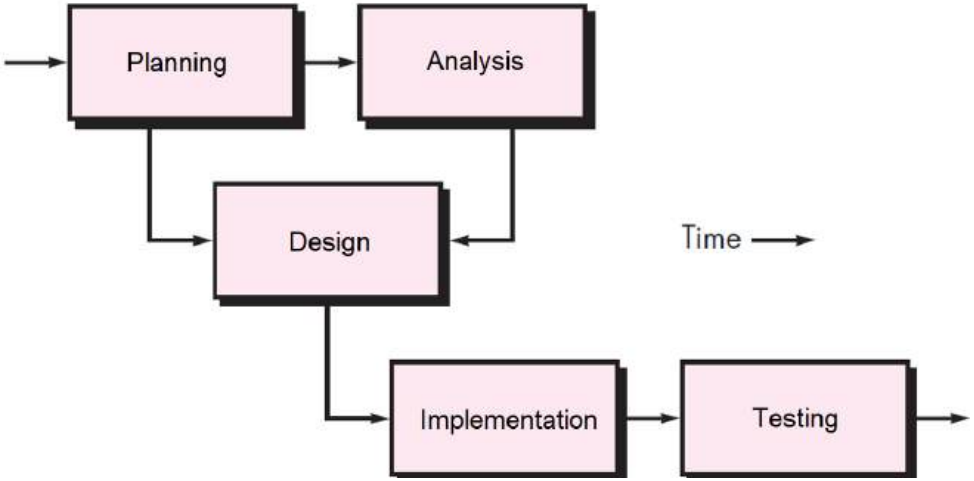
(b) Iterative process flow

Process Flow



(c) Evolutionary process flow

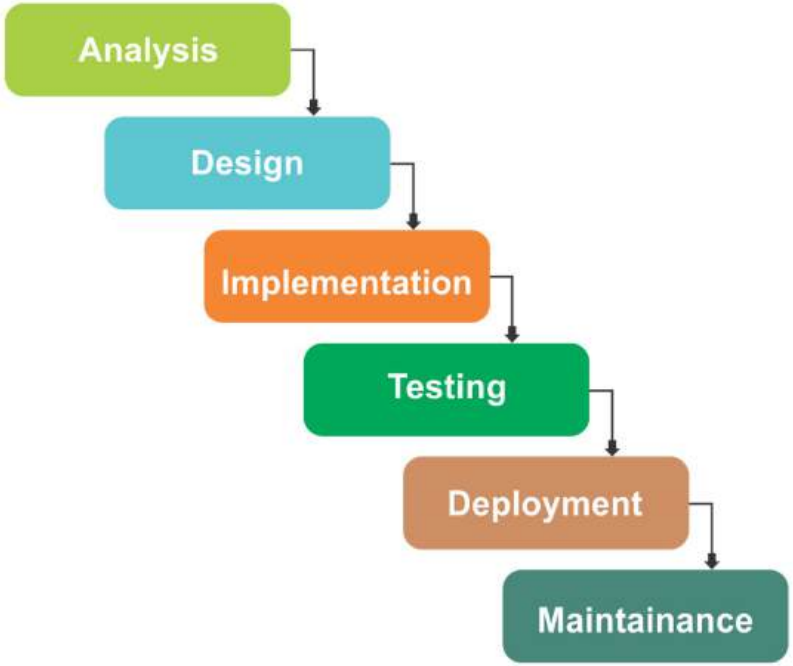
Process Flow



(d) Parallel process flow

Model *Waterfall* (Royce, 1970)

PROGRAM STUDI
TEKNOLOGI REKAYASA
PERANGKAT LUNAK
POLIBATAM

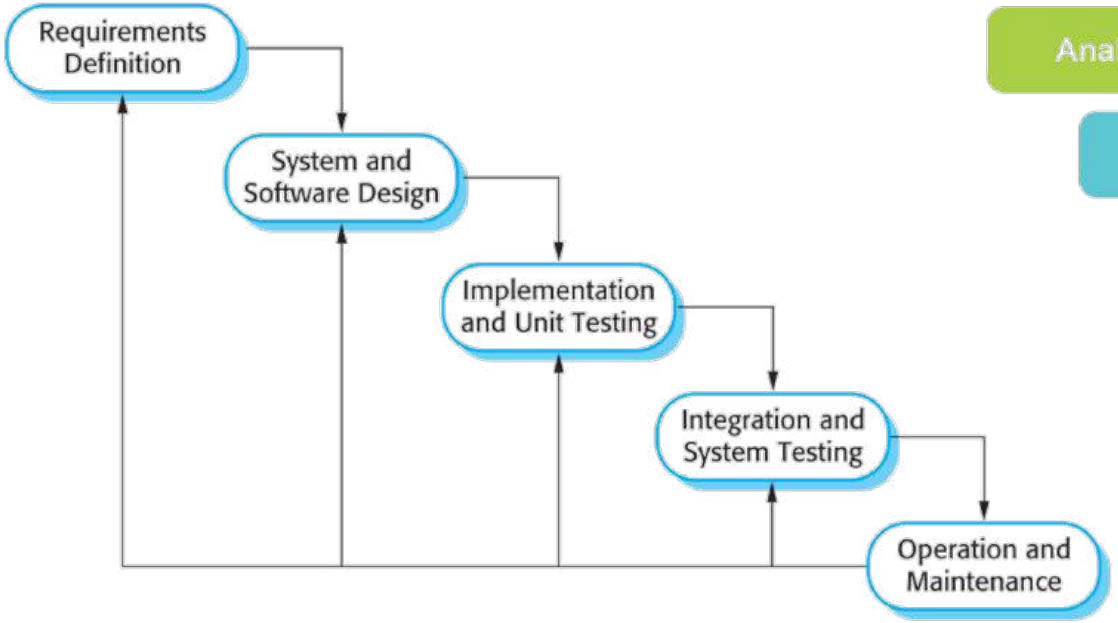


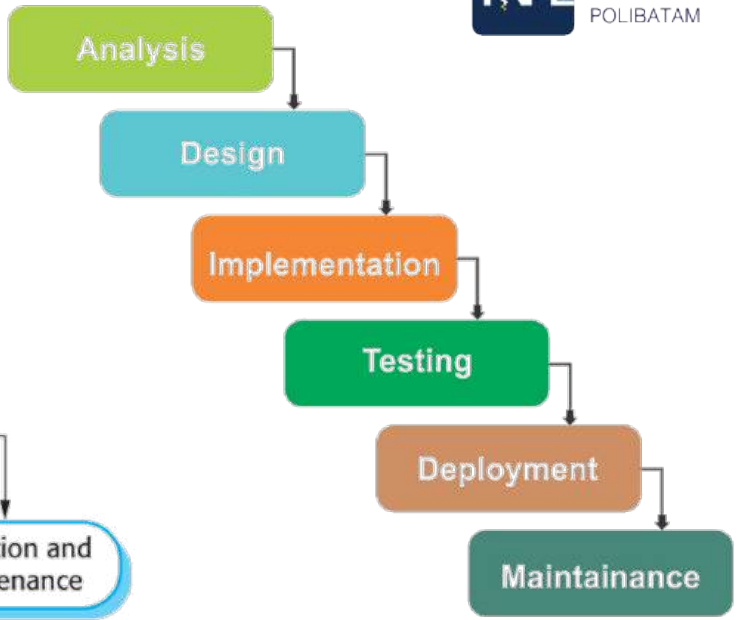
Sebuah tahap harus diselesaikan sebelum maju ke tahap berikutnya

Model ini hanya sesuai pada software dengan requirements yang sudah jelas terdefinisi dan tidak banyak perubahan pada proses desain

Kelemahan:

- Tidak mudah bagi customer untuk mendefinisikan semua requirements
- Versi software yang dapat diajalankan baru akan terbangun di tahap akhir proyek



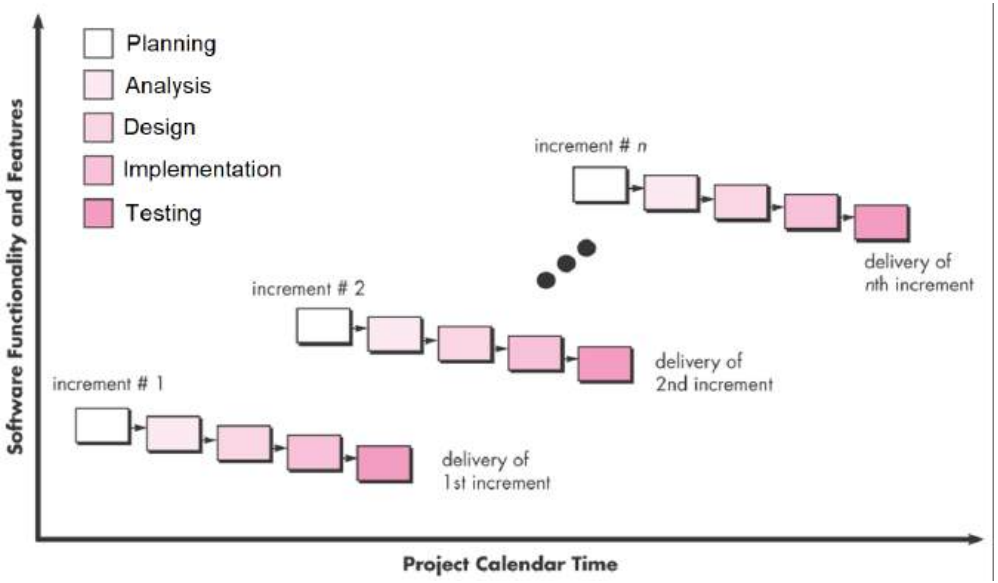


Walaupun dalam versi asli model waterfall yang ditemukan oleh Winston Royce (1970) terdapat “feedback loops” (gambar kiri), namun biasanya implementasi dari model ini dilakukan secara linear saja (gambar kanan)

Incremental Process Model

Membangun versi awal, kemudian menerima komentar dari user dan mengubah versi awal tersebut sampai selesai

Setiap versi menambahkan fungsi yang dibutuhkan user

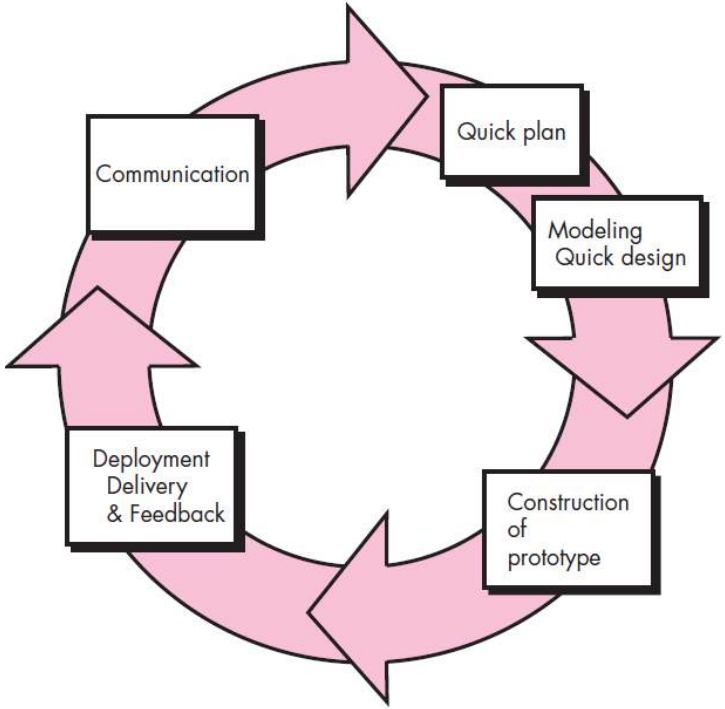


Prototyping

Prototype adalah versi awal sebuah sistem yang digunakan untuk mendemonstrasikan konsep dan mencoba berbagai pilihan desain, dan mendalami masalah serta pemecahannya

Dalam beberapa kasus, *prototype* ini tidak dapat digunakan lagi sehingga untuk sistem yang sebenarnya, developer membuat kembali dari awal ⇨ '*throw-away prototypes*'

Prototype juga dapat bersifat evolutionary dimana *prototype* akan terus dikembangkan/berevolusi menjadi produk yang sebenarnya

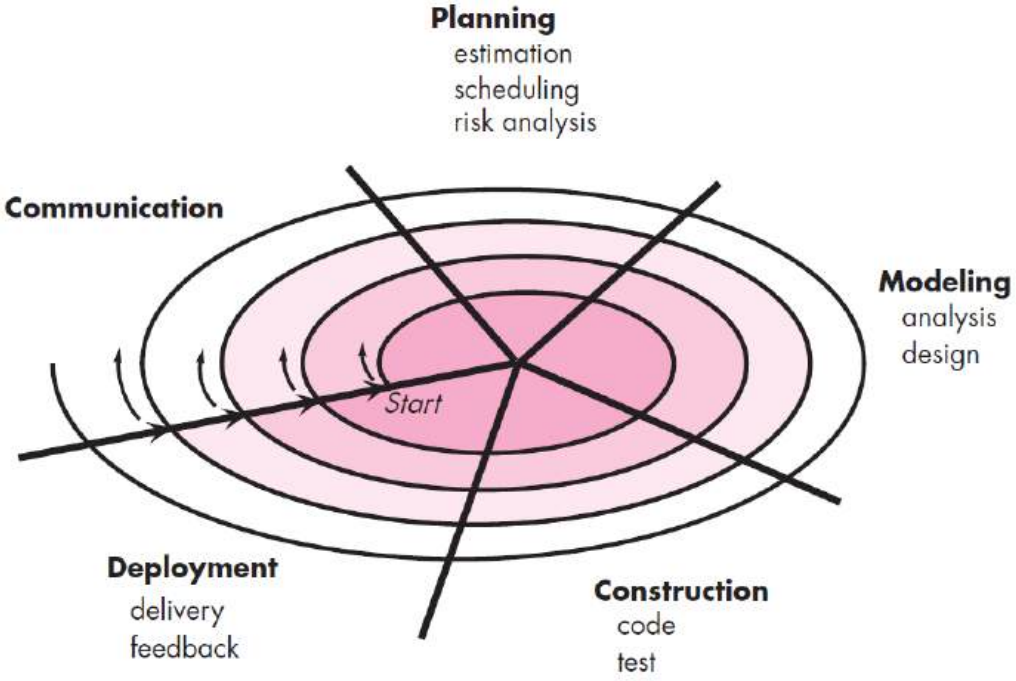


Prototyping

Kelemahan:

- Sebagai pengembang perangkat lunak, kita seringkali melakukan beberapa kompromi dalam implementasi untuk mendapatkan prototipe secara cepat.
- Customer melihat perangkat lunak yang terlihat sudah jadi dan berjalan, padahal prototype biasanya dibuat sejadinya, tanpa memperhitungkan kualitas secara keseluruhan atau aspek kinerja perangkat lunak yang lain secara jangka panjang.

The Spiral Model (Barry Boehm, 1998)



Menggabungkan sifat iterative dari prototyping dengan aspek sistematis dari model waterfall

Merupakan model proses bebasis risiko yang digunakan untuk memandu pembangunan perangkat lunak

The Spiral Model

PROGRAM STUDI
TEKNOLOGI REKAYASA
PERANGKAT LUNAK
POLIBATAM

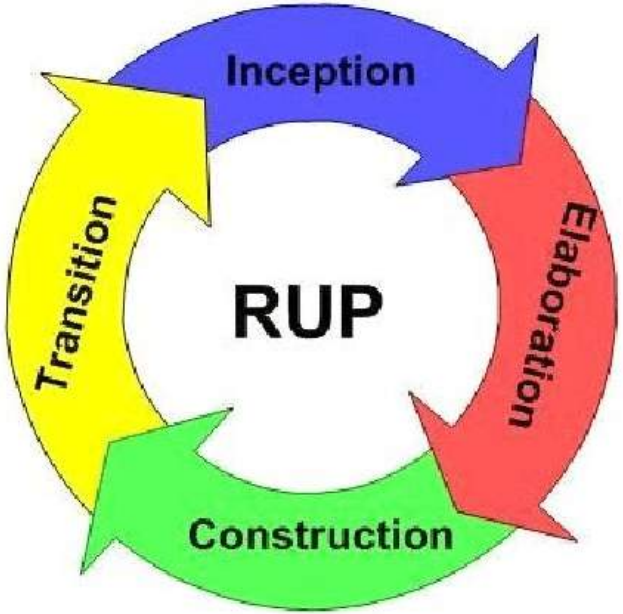
Dengan spiral model, perangkat lunak dikembangkan dengan serangkaian rilis yang bersifat evolutionary

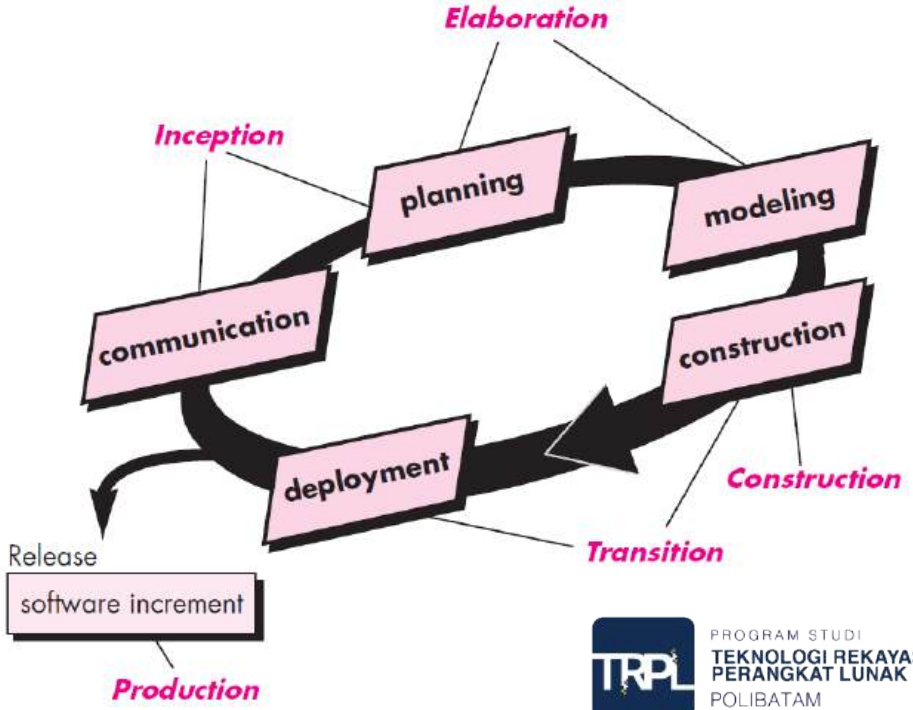
- Dalam iterasi-iterasi awal, rilis dapat berupa sebuah model atau prototype. Dalam iterasi-iterasi berikutnya, dihasilkan versi perangkat lunak yang lebih lengkap.

Kelemahan:

- Sulit untuk meyakinkan customers (terutama terkait kontrak) bahwa pendekatan yang bersifat evolutionary dapat dikendalikan dengan baik.
- Diperlukan keahlian dalam manajemen risiko dan kesuksesan penggunaan spiral model sangat bergantung pada keahlian tersebut. Jika ada risiko besar yang tidak terungkap dan tidak dikelola dengan baik, maka pasti akan muncul masalah.

Rational Unified Process (RUP)
(James Rumbaugh, Grady Booch, Ivar Jacobson, 1999)





PROGRAM STUDI
TEKNOLOGI REKAYASA
PERANGKAT LUNAK
POLIBATAM

Rational Unified Process (RUP)

PROGRAM STUDI
TEKNOLOGI REKAYASA
PERANGKAT LUNAK
POLIBATAM

Fase *inception*—berkolaborasi dengan stakeholder, kebutuhan perangkat lunak didefinisikan; arsitektur kasar sistem dibuat; dan rencana proyek yang bersifat iterative dan incremental dikembangkan.

Fase *elaboration*—memperbaiki dan mengembangkan lebih jauh use case serta gambaran arsitektur

Fase *construction*—mengembangkan atau mendapatkan berbagai komponen perangkat lunak yang akan digunakan untuk membangun use case sehingga dapat digunakan oleh pengguna

Fase *transition*—tahapan akhir dari aktivitas construction dan bagian awal dari aktivitas deployment (delivery dan feedback).

Fase *production*—penggunaan perangkat lunak dimonitor, support terhadap lingkungan operasional perangkat lunak disediakan, dan laporan megenai kerusakan dan permintaan akan perubahan disubmit dan dievaluasi.

Specialized Process Model

Component-Based Development—membangun aplikasi dari komponen perangkat lunak yang sudah ada

Formal Methods Model—perangkat lunak dimodelkan menggunakan spesifikasi formal matematika. Variasi dari pendekatan ini disebut cleanroom software engineering

Aspect-Oriented Software Development—pendekatan untuk mendenisikan, menspesifikasikan, merancang, dan membangun aspect. Aspect mendefinisikan crosscutting concerns memiliki pengaruh terhadap keseluruhan arsitektur perangkat lunak. Crosscutting concerns merupakan bagian yang ada atau tersebar dalam banyak fungsi, fitur dan informasi sistem, misalnya keamanan. Concern merupakan kebutuhan dari customer atau sisi teknis yang diperlukan.

Model Proses Agile
Agile Process Model

Software Process

Pada tahun 2001, Kent Beck dan 16 orang software developer, penulis, dan konsultan ternama (yang dikenal sebagai “Agile Alliance”) mencanangkan “Manifesto for Agile Software Development.”

Nilai-nilai dasar agile manifesto tersebut adalah:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile Principles

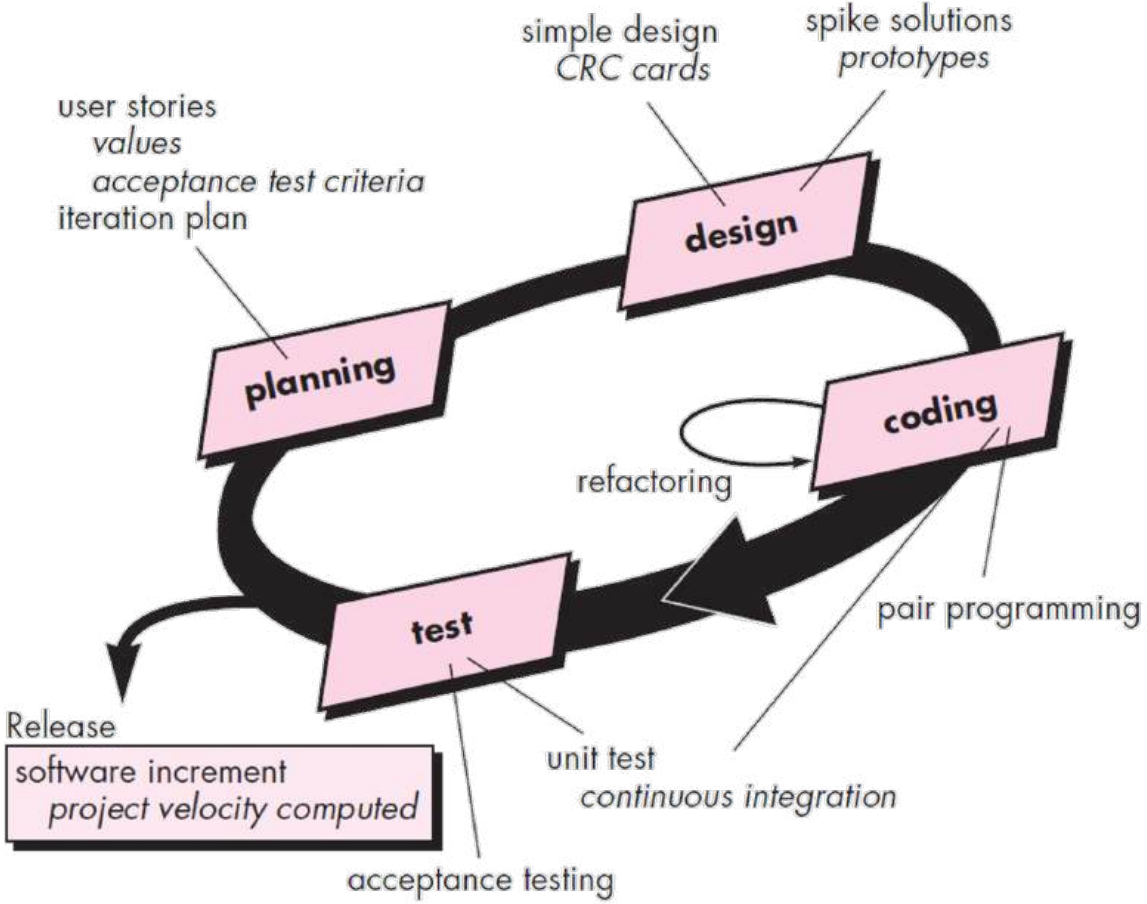
PROGRAM STUDI
TEKNOLOGI REKAYASA
PERANGKAT LUNAK
POLIBATAM

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile Process Models

- Extreme Programming
- Scrum
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Crystal
- Feature Drive Development (FDD)
- Lean Software Development (LSD)
- Agile Modeling (AM)
- Agile Unified Process (AUP)

Extreme Programming



Extreme Programming

Requirement dituliskan sebagai scenario (disebut *user stories*), yang diimplementasikan langsung sebagai serangkaian aksi.

Programmer bekerja secara berpasangan (*pair programming*) dan membangun pengujian untuk setiap aksi sebalum menuliskan code. Semua pengujian harus berhasil dieksekusi saat code yang baru diintegrasikan dalam sistem.

Versi software yang baru dapat dihasilkan beberapa kali dalam sehari dan rilis diberikan kepada customer sekitar dua minggu sekali.

'*Spike*', sebuah penambahan namun tidak melakukan programming apapun, misalnya: pengembangan untuk memahami persoalan, membuat dokumentasi sistem, dsb.

Contoh ‘user story’

Prescribing Medication

Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select current medication’, ‘new medication’ or ‘formulary’.

If she selects ‘current medication’, the system asks her to check the dose. If she wants to change the dose, she enters the dose and then confirms the prescription.

If she chooses ‘new medication’, the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check that the medication selected is correct. She enters the dose and then confirms the prescription.

If she chooses ‘formulary’, the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose and then confirms the prescription.

The system always checks that the dose is within the approved range. If it isn’t, Kate is asked to change the dose.

After Kate has confirmed the prescription, it will be displayed for checking. She either clicks ‘OK’ or ‘Change’. If she clicks ‘OK’, the prescription is recorded on the audit database. If she clicks on ‘Change’, she reenters the ‘Prescribing medication’ process.

CRC/task card untuk
'*Prescribing Medication*'

Task 1: Change Dose of Prescribed Drug

Task 2: Formulary Selection

Task 3: Dose Checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose.

Using the formulary ID for the generic drug name, look up the formulary and retrieve the recommended maximum and minimum dose.

Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low.

If within the range, enable the 'Confirm' button.

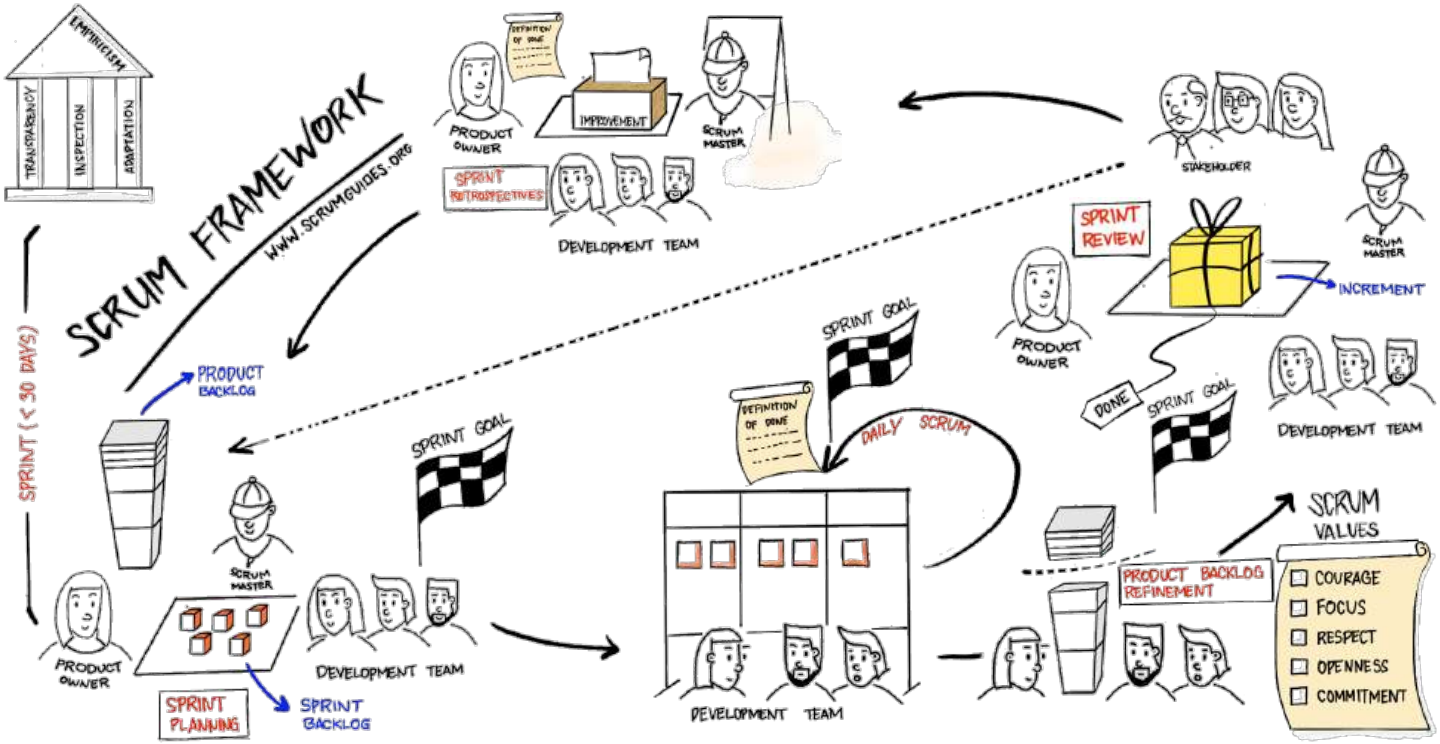
Deskripsi kasus uji untuk task '*Dose Checking*'

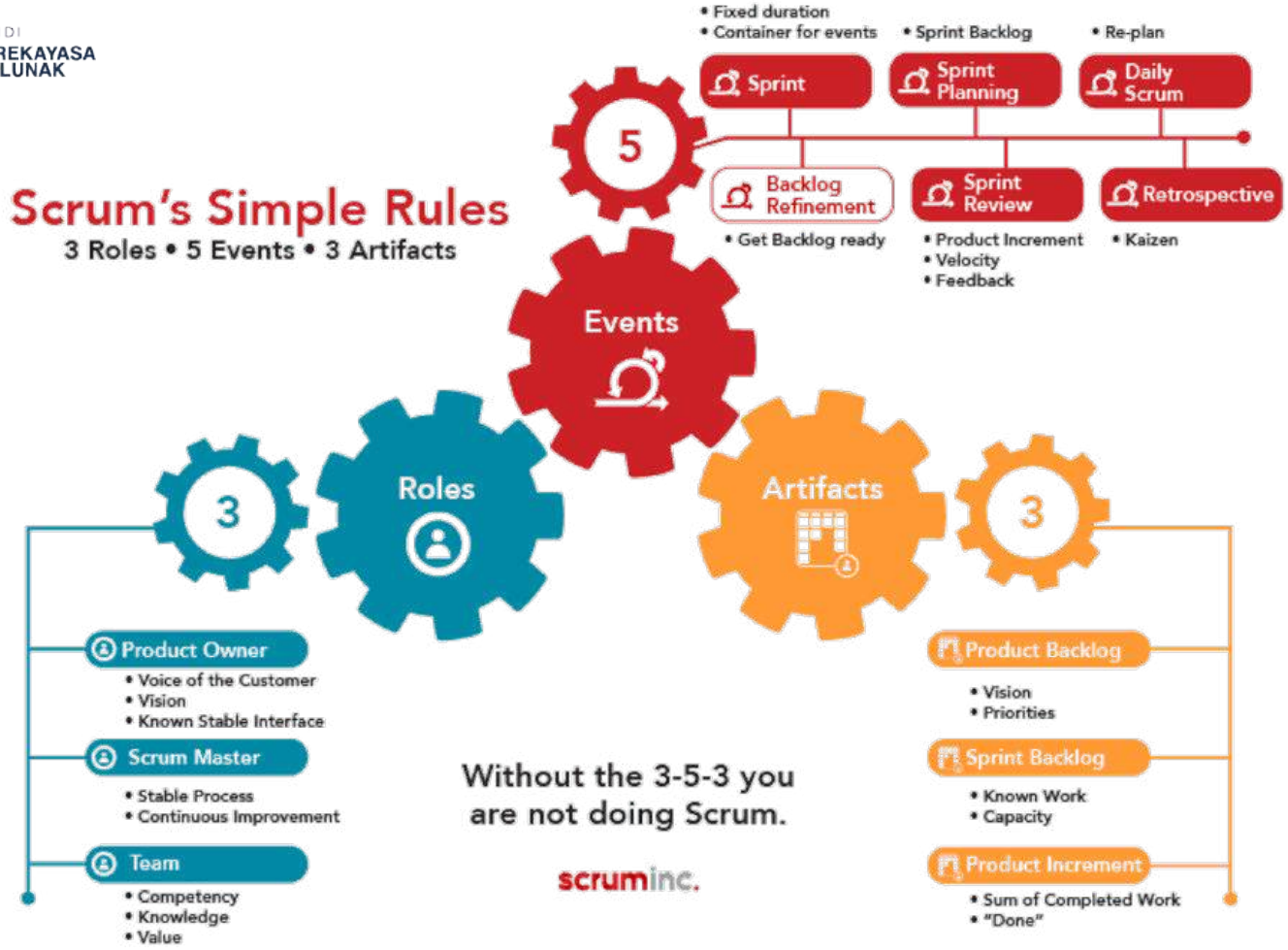
Test 4: Dose Checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

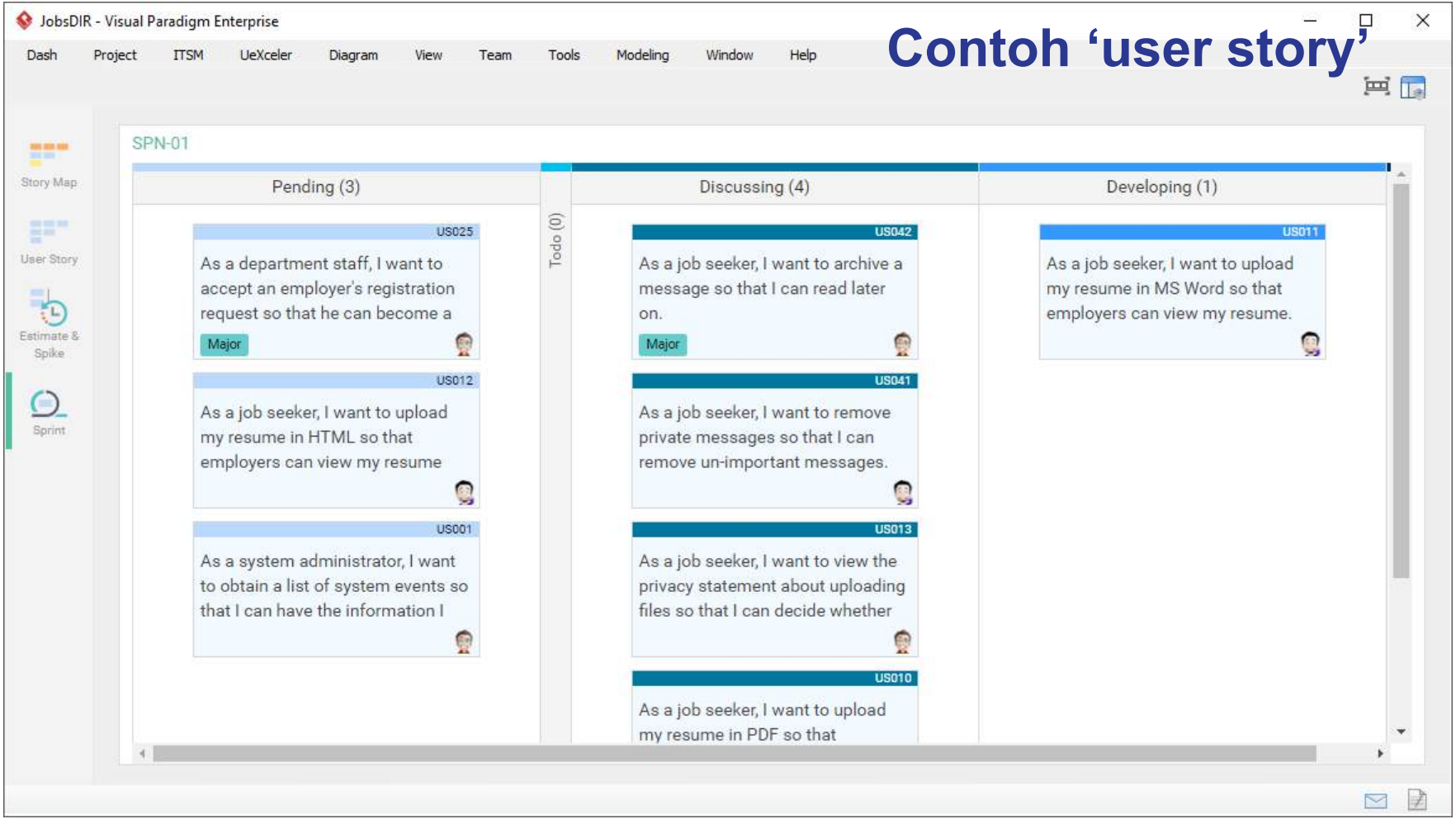
Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose × frequency is too high and too low.
4. Test for inputs where single dose × frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.


EMPIRICISM
TRANSPARENCY
INSPECTION
ADAPTATION
SCRUM FRAMEWORK
WWW.SCRUMGUIDES.ORG
SPRINT (< 30 DAYS)
PRODUCT OWNER
DEFINITION OF DONE
IMPROVEMENT
SCRUM MASTER
SPRINT RETROSPECTIVE
DEVELOPMENT TEAM
STAKEHOLDER
SPRINT REVIEW
INCREMENT
PRODUCT BACKLOG
SPRINT GOAL
DAILY SCRUM
DONE
SPRINT PLANNING
SPRINT BACKLOG
PRODUCT BACKLOG REFINEMENT
SCRUM VALUES
COURAGE
FOCUS
RESPECT
OPENNESS
COMMITMENT


Scrum's Simple Rules
3 Roles • 5 Events • 3 Artifacts
5
Events
• Fixed duration
• Container for events
Sprint
• Sprint Backlog
Sprint Planning
• Re-plan
Daily Scrum
Backlog Refinement
• Get Backlog ready
Sprint Review
• Product Increment
• Velocity
• Feedback
Retrospective
• Kaizen
3
Roles
Product Owner
• Voice of the Customer
• Vision
• Known Stable Interface
Scrum Master
• Stable Process
• Continuous Improvement
Team
• Competency
• Knowledge
• Value
Artifacts
3
Product Backlog
• Vision
• Priorities
Sprint Backlog
• Known Work
• Capacity
Product Increment
• Sum of Completed Work
• "Done"
Without the 3-5-3 you are not doing Scrum.
scruminc.

Contoh ‘user story’

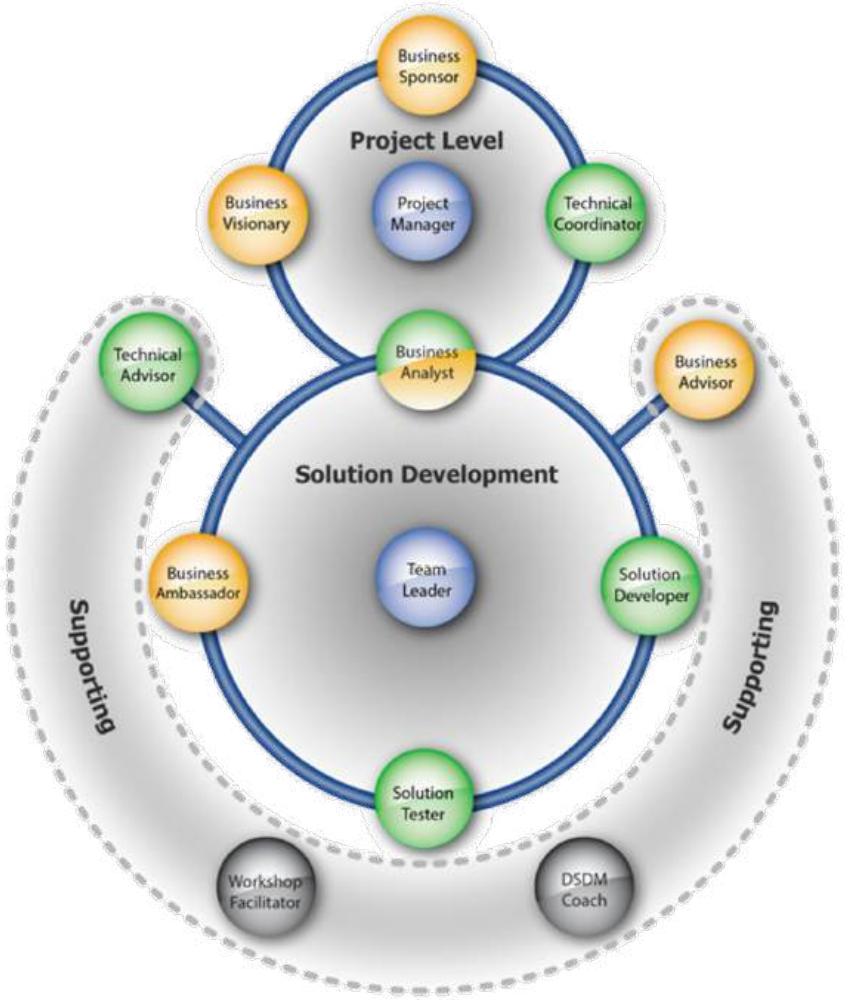


Dynamic Systems Development Method (DSDM)

Sebuah metode dan framework pengelolaan proyek agile yang iterative, incremental dan adaptive (change-driven/empirical) dan dimaksudkan untuk proyek pengembangan perangkat lunak (namun dapat digunakan dalam ranah bisnis lain).

Metode ini memiliki sejarah yang panjang – 20 tahun dengan 6 rilis/versi, menjadikan metode ini merupakan salah satu metode pengelolaan proyek agile tertua.

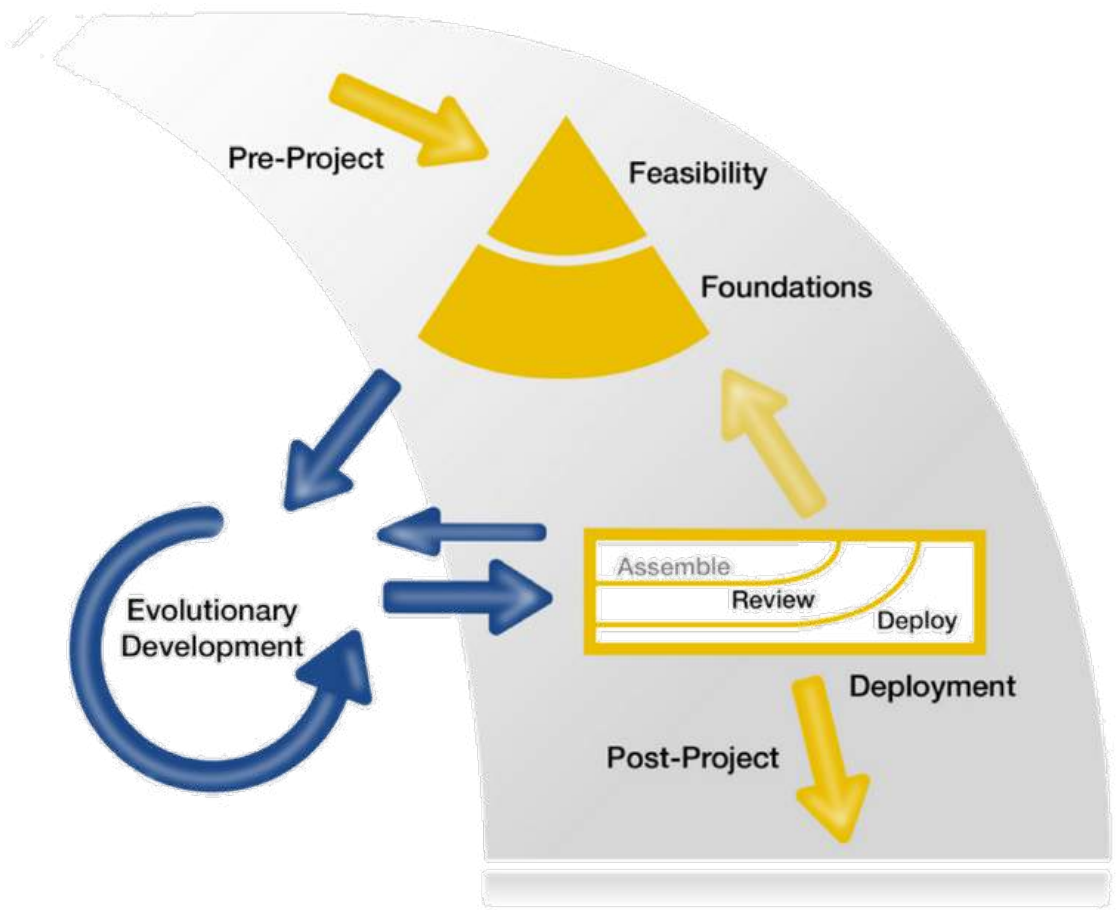
DSDM® dipandang sebagai metode hybrid, yang mengkombinasikan pengelolaan proyek dengan pengembangan produk dalam satu daur hidup dan metode.



DRDM Roles (13)

(source: DSDM® AgilePF® - Agile Project Framework (6th version / 2014)

PROGRAM STUDI
TEKNOLOGI REKAYASA PERANGKAT LUNAK
POLIBATAM



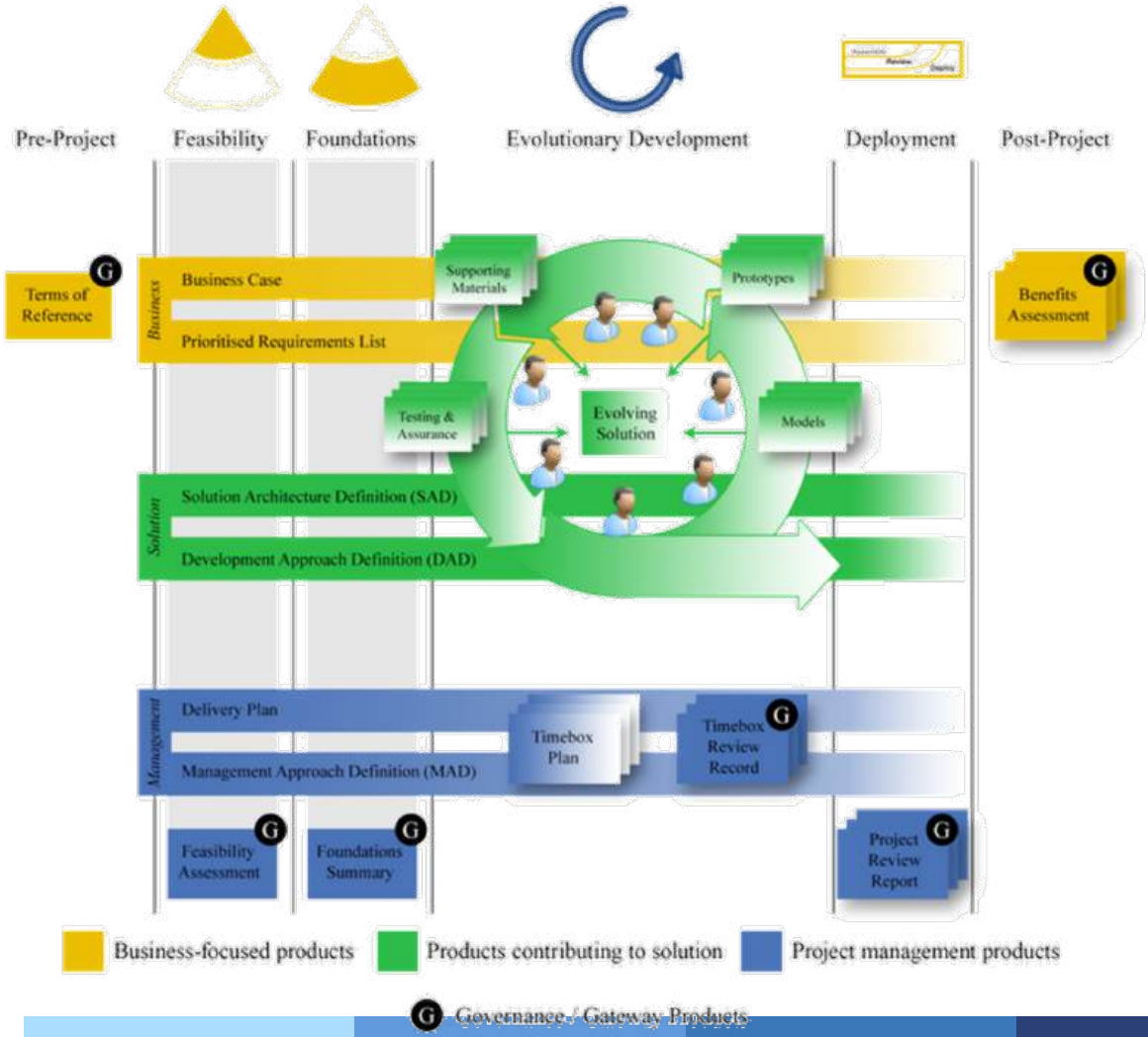
DRDM Process (1) with Phases (6)

(source: DSDM® AgilePF® - Agile Project Framework (6th version / 2014)

PROGRAM STUDI
TEKNOLOGI REKAYASA
PERANGKAT LUNAK
POLIBATAM

DRDM Products/Artifacts (14)

(source: DSDM® AgilePF® - Agile Project Framework (6th version / 2014)



Feature Driven Development (FDD)

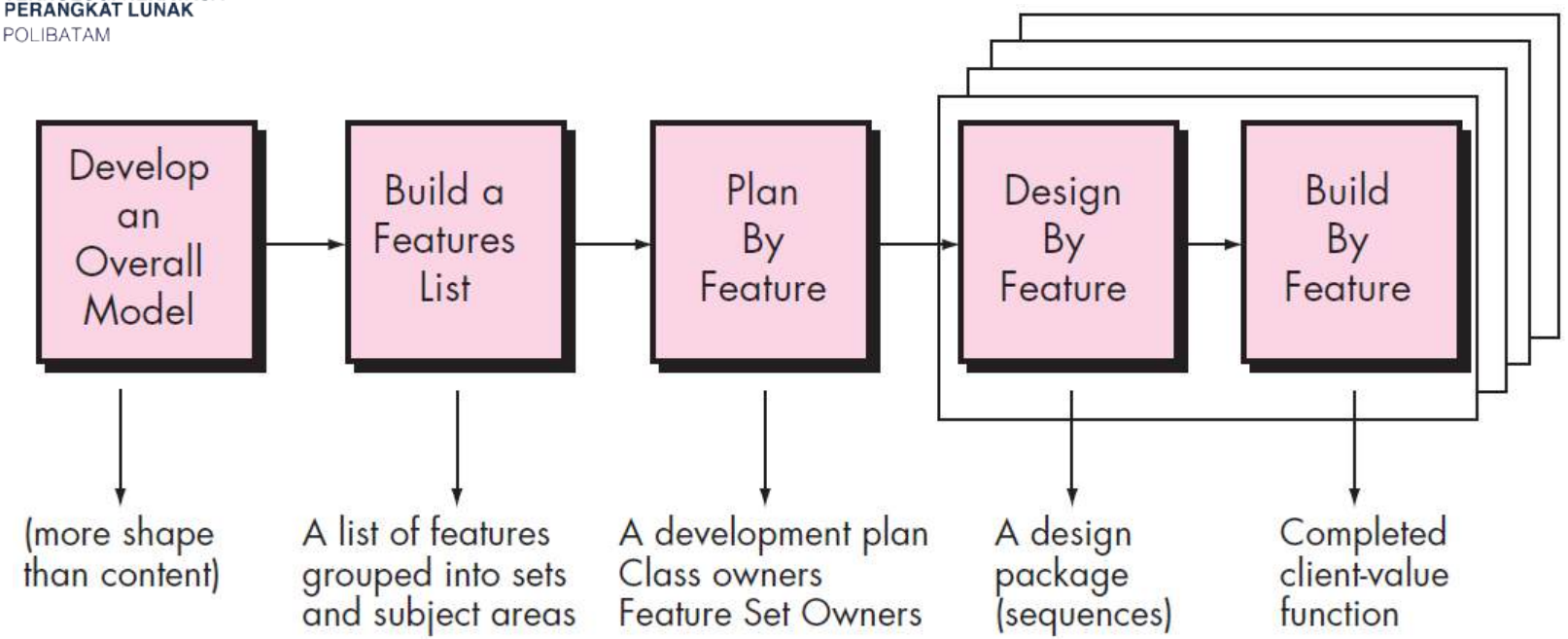
Peter Coad dkk (1997): FFD merupakan proses model yang praktis untuk rekayasa perangkat lunak berorientasi objek.

Stephen Palmer dan John Felsing (2002): menambahkan kepada hasil kerja Coad, mendeskripsikan proses yang adaptif dan agile yang dapat diaplikasikan untuk proyek software berukuran menengah hingga besar.

Sebuah feature merupakan fungsi yang dibutuhkan client yang dapat diimplementasikan dalam waktu dua minggu atau kurang (Coad, 1997); dalam bentuk

<action> the <result> <by for of to> a(n) <object>

Contoh: Tambahkan produk ke keranjang belanja; Tampilkan spesifikasi teknis dari sebuah produk; Simpan informasi belanja untuk seorang customer; dsb.

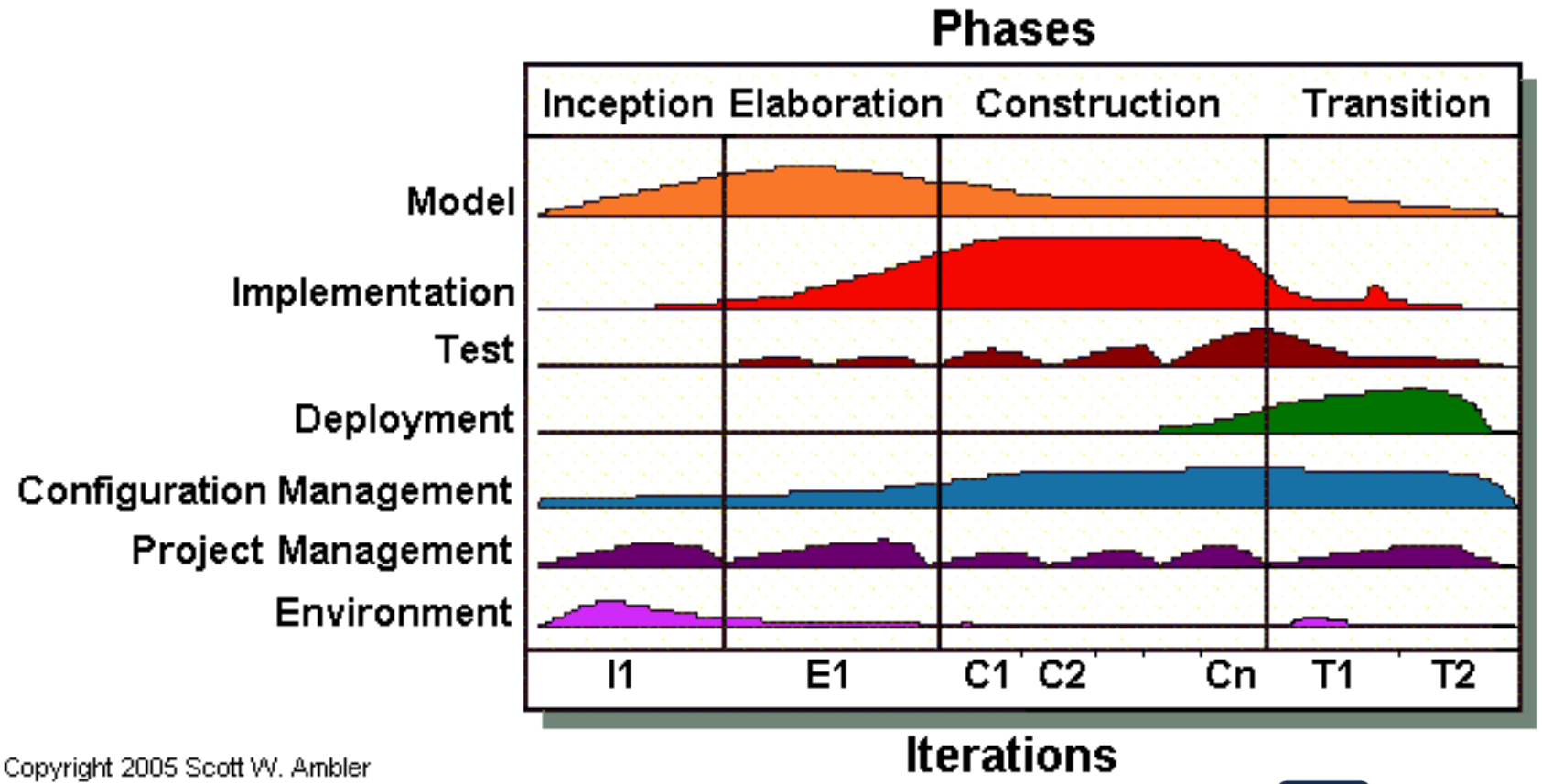


Feature Driven Development (Coad, 1997)

Agile Unified Process (AUP)

Bentuk sederhana dari Rational Unified Process (RUP) yang dibuat oleh Scott Ambler (2005)

Mengadopsi aktivitas klasik UP—inception, elaboration, construction, dan transition—namun dalam setiap aktivitas, tim melakukan iterasi untuk agar agile dan dapat menghasilkan software kepada pengguna secepat mungkin



Copyright 2005 Scott W. Ambler

AUP Activities

- *Modeling*. UML menggambarkan domain bisnis dan masalah yang dibangun. Namun, agar tetap “agile”, model tersebut hanya “cukup baik” agar tim dapat bekerja.
- *Implementation*. Model diterjemahkan menjadi source code.
- *Testing*. Seperti XP, tim merancang dan menjalankan serangkaian pengujian untuk menemukan error dan memastikan bahwa source code memenuhi requirement.
- *Deployment*. Seperti aktivitas biasanya, deployment berfokus pada penyerahan software dan mendapatkan feedback dari pengguna.
- *Configuration and project management*. Dalam AUP, configuration management melakukan pengelolaan perubahan, pengelolaan risiko, dan pengendalian berbagai produk yang dihasilkan. Pengelolaan proyek melacak dan mengendalikan kemajuan tim dan mengkoodinasikan aktivitas tim.
- *Environment management*. Environment management mengkoordinasi berbagai infrstruktur termasuk standard, tool, dan berbagai teknologi yang disediakan untuk tim.

Selesai. ☺