Hugbúnaðarverkefni 1 / Software Project 1

2. Rational Unified Process

HBV501G – Fall 2018

Matthias Book

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

In-Class Quiz Prep

- Please prepare a scrap of paper with the following information:

- ID: ____@hi.is Date: ________

- a) __________ e) __________
- b) __________ f) __________
- c) __________ g) __________
- d) __________ h) __________

- During class, I'll show you questions that you can answer with numbers
  - No further elaboration necessary
- Hand in your scrap at end of class

- All questions in a quiz weigh same
- All quizzes (8-10 throughout semester) have the same weight
  - Your worst 2 quizzes will be disregarded
- Overall quiz grade counts as optional question worth 7.5% on final exam

Course Advertisement

Security Requirements Engineering for Cyber-Physical Systems (TÖL505M)

- 2-credit course for Bachelor and Master students
- 4 weeks (10 Sep – 5 Oct), 4 lectures, 2 assignments
  - Tentative schedule (likely subject to change):
    Wednesdays 10:00-12:20 & Fridays 12:30-13:10 @ Interaction Room (Tæknigarður 227)
- Lecturer: Shafiq ur Rehman, MSc (University of Duisburg-Essen, Germany)
  - Contact: shafiq@hi.is, book@hi.is

- **Contents:**
  - Basic definitions of security threats, security goals and risk assessment
  - Overview of concepts associated with industrial cybersecurity
  - Main security threats to software development lifecycle
  - Types of attacks on critical infrastructure
  - Main standards and methodologies of security requirements engineering

Team Consultations

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
A	X			X			X			X			X							X			X			X				
D		X			X			X			X					X					X			X			X			
M			X			X			X			X			X				X						X			X		X
#								2					2	0			0	0	2			0					2		0	

- **Consultation time slots**
  - Thursday 08:30-11:30, V-152
    - Teams 1-10: 08:30-09:30
    - Teams 11-20: 09:30-10:30
    - Teams 21-30: 10:30-11:30
  - Seating order
    - Andri's teams: last two rows
    - Daníel's teams: middle row(s)
    - Matthias' teams: first two rows

- **Team and topic finalization**
  - Fix team members in Doodle by **Sun 9 Sep**
  - Fix project idea with your team by **Wed 12 Sep**
  - Attend consultations on **Thu 13 Sep**
    - to finalize idea with tutor
    - to define project scope

Software Process Models

see also:

Pressman: Software Engineering – A Practitioner’s Approach, Ch. 1 & 2

The Nature of Software Development

**“Because software is embodied knowledge,
and that knowledge is initially dispersed, tacit, latent, and incomplete,
software development is a social learning process.”**

Howard Baetjer, Jr.: Software as Capital. IEEE Computer Society Press, 1998

The Nature of Software

- Software comprises:
  - **Programs** (sets of computer instructions) that provide desired features when executed
  - **Data structures** that enable programs to adequately manipulate information
  - **Documentation** that describes the operation and use of programs

- Software vs. physical products
  - Software is developed or engineered, it is not manufactured.
  - Software does not “wear out”, but it can deteriorate over time.
  - Software is mostly custom-built (despite a trend toward component-/service-based systems)

- **Software is both a product and a vehicle that delivers a product.**

Types of Software
(Categories are not disjoint, but characterized by specific challenges)

- **System software**
  - Heavy interaction with hardware, no dedicated user interface (e.g. printer drivers, …)
- **Application software**
  - Stand-alone software addressing particular need (e.g. desktop apps, scientific computing, …)
- **Distributed systems**
  - Code and data spread across multiple hardware devices (e.g. web applications, agents, …)
- **Embedded systems**
  - Highly specialized, highly hardware-dependent, real-time tasks (e.g. cruise control, …)
- **Product-line software**
  - Many variation points to serve individual users' needs (e.g. automotive engine control, …)
- **Games**
  - Heavy focus on user experience, often with real-time aspects (e.g. arcade, first-person, …)
- **Artificial intelligence systems**
  - Techniques such as machine learning, neural networks etc. (e.g. pattern recognition, …)

Software as a Product and a Vehicle Delivering a Product

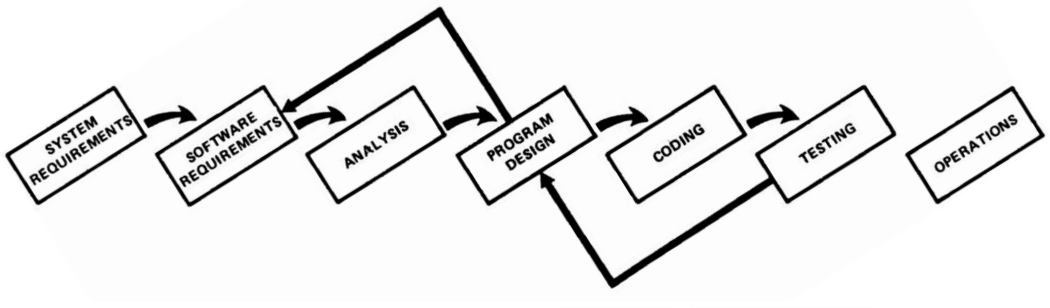
- There are many different [opinions on] things that software could and should do.
  - Developers need to understand the problem **(requirements)** before developing a solution.
- Application domains and technical toolsets are highly complex.
  - Smooth interaction of all software components requires careful consideration **(design)**.
- Software errors have (possibly dramatic) consequences in real life.
  - Developers must ensure their software exhibits high quality **(testing)**.
- Successful software will be used and built on for a long time.
  - Software should be built to be adaptable and extensible **(maintenance)**.

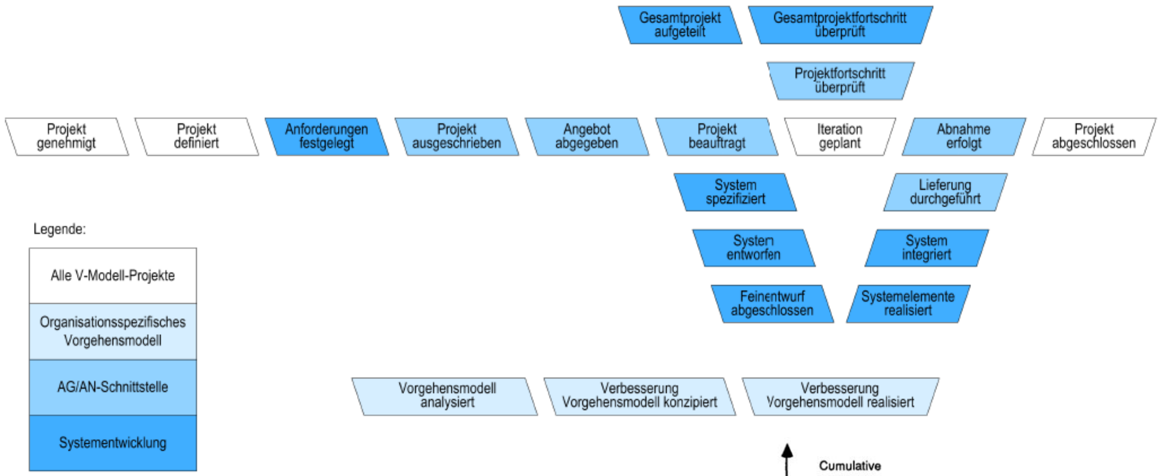
Software Engineering Activities

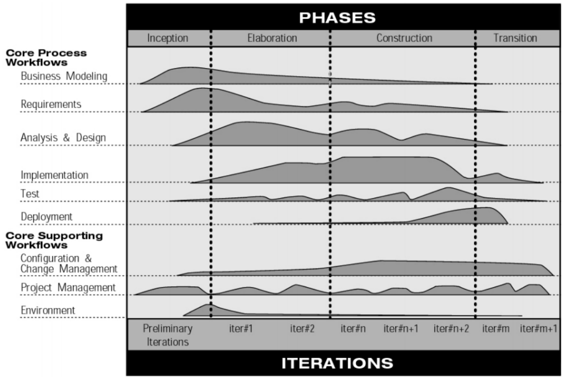
- **General Activities:**
  - Communication – understand what the stakeholders need
  - Planning – consider how you will build a solution in the given time / budget
  - Design – figure out how exactly the solution should work
  - Construction – build the solution according to your design
  - Deployment – roll out the solution for use by the stakeholders
  - [Maintenance – all of the above, on a smaller, more detail-focused scale]
- **Umbrella Activities:**
  - Quality assurance – make sure you build the right solution, the right way
  - Project management – allocate resources, track progress
  - Risk management – identify and eliminate risks
  - Process improvement – measure/review how you are doing, learn, improve your methods

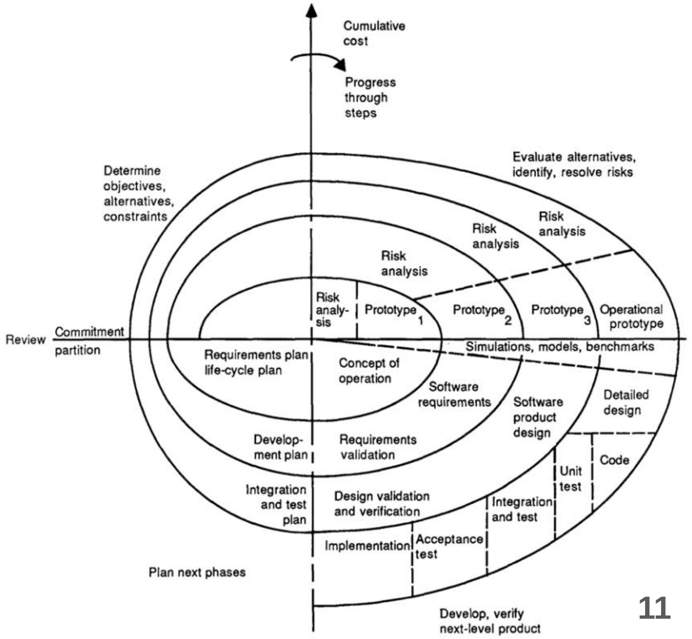
Software Process Models

- A **software process model** defines
  - how the various software engineering activities are structured in time
  - how they depend on each other through the creation and consumption of artifacts
- Large variety of models
  - Waterfall model, V model, spiral model, Unified Process (UP), agile models…
- Major differences
  - Structure vs. flexibility
  - Emphasis placed on different activities and artifacts









Issues with Sequential Process Models

- A sequential model (i.e. an extreme version of the “waterfall”) attempts to
  - identify all or most requirements at the start of the project
  - create a thorough design as a “blueprint” of the complete system before programming starts
  - defines a complete project schedule at the beginning of the project
- This assumes that
  - no requirements will change over the course of the project
  - all implementation details can be foreseen at the start of the project
  - all initial time and effort estimates are accurate
  - no unforeseen questions, uncertainties, conflicts etc. come up
  - developers have the discipline to follow all specifications and schedules precisely
- Obviously unrealistic: The larger the project, the less true these assumptions.
  - Caution: Nevertheless, it is very tempting to plan a project under these assumptions!
    - Double-check: Do your “iterations” actually mirror the steps of a strict waterfall model?

Iterative-Incremental Process Models

- Split the project into **iterations**
- Each iteration is a mini-project…
  - Understand requirements
  - Design, code and test software
- …and produces an **increment**, i.e. an executable partial system
  - Basis for review and feedback
  - Foundation for next iteration

- **Benefits:**
  - Better productivity, lower defect rates
  - Earlier identification, mitigation of risks
  - Early visible progress
  - Early user engagement and feedback, i.e. closer alignment with user needs
  - Manageable complexity in each iteration
  - Lessons learned in one iteration can help to improve the next

In Defense of Plan-Driven Process Models

- Some agile proponents mistakenly equate all plan-driven models with strict sequential models (“if it’s not agile, it’s waterfall!”).
- Actually, all modern plan-driven models are iterative-incremental.
  - (if they are applied properly and do not get a “strict waterfall” superimposed on them)
- Difference is in:
  - the necessary amount of pre-planning
  - the necessary amount of structured design
  - the certainty of the overall target vision
- …in each iteration, depending on:
  - the criticality of the application domain (e.g. health/financial risks)
  - the complexity of the application domain, technology, existing system landscape

Plan-driven vs. Agile Iterative Development

Plan-driven Iterative Development

- Gain understanding through models and specifications
- Risk management through planning
- Aspiration for stable structures
- Optimization through planning
- Work on most risky features first
- Increments are partial systems
- Stable overall target vision
- Well-defined roles and responsibilities
- Discipline required to follow plans

Agile Iterative Development

- Gain understanding through communication and feedback
- Risk management through flexibility
- Acceptance of fluid structures
- Optimization through refactoring
- Work on most valuable features first
- Increments should be viable products
- Open overall target vision
- Self-organizing teams
- Discipline required to utilize freedom

Plan-driven vs. Agile Iterative Development

Plan-driven Iterative Development

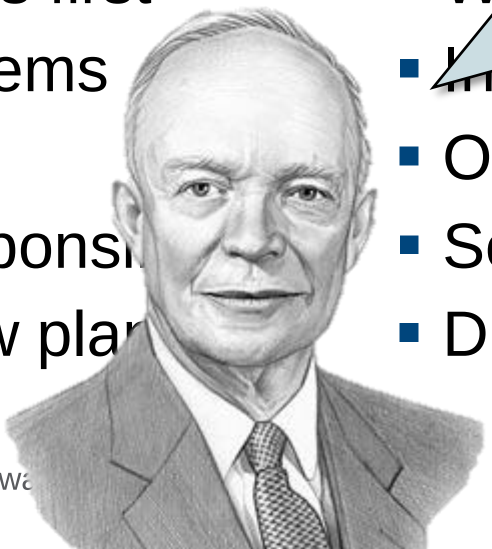
- Gain understandi
  models and spec
- Risk managemen
- Aspiration for stal
- Optimization thro
- Work on most risky features first
- Increments are partial systems
- Stable overall target vision
- Well-defined roles and responsi
- Discipline required to follow plan

Agile Iterative Development

- ough
  dback
- igh flexibility
- ctures
- factoring
- Wo st valuable features first
- Increments should be viable products
- Open overall target vision
- Self-organizing teams
- Discipline required to utilize freedom

In preparing for battle,
I have always found that plans are useless,
but planning is indispensable.

-- Dwight D. Eisenhower



Expectation Management: Education vs. Practice

- Your projects in this course will likely be relatively simple
  - given the timeframe of three months until delivery
  - given the effort you will be able to invest beside your other coursework
  - given the synchronization issues of learning about methods in parallel to applying them
- In industrial practice, this simplicity would make them obvious candidates for
  - an agile development process
  - a client-side web technology
- However, our learning goal is to understand how to use
  - a plan-driven process
  - object-oriented design
- Initial experience with these is best gained using a lightweight project example.
  - (Plus, complexity of the technology should motivate some of the complexity of the process.)

The Unified Process

see also:

- Larman: Applying UML and Patterns, Ch. 2

The Unified Process (UP)

- Developed by Ivar Jacobson, Grady Booch and James Rumbaugh in parallel with the Unified Modeling Language (UML); first published in 1999
- Defines a number of roles, workflows, activities and artifacts
  - See http://sce.uhcl.edu/helm/rationalunifiedprocess/ for a browseable reference
  - Most of these are optional
  - UP needs to be tailored to individual project situations
    - Make deliberate decisions on which elements you need and why!
- Many refinements and variations proposed over time, e.g.:
  - **Rational Unified Process (RUP) – IBM's version**
  - Oracle Unified Method (OUM) – Oracle's version
  - Open Unified Process (OpenUP) – the Eclipse Foundation's version
  - Essential Unified Process (EssUP) – Jacobson's lightweight version

Unified Process Structure

- The UP defines four **phases** (inception, elaboration, construction, transition).
  - Each phase produces a **milestone** composed of a set of artifacts.
- Each phase consists of several **iterations**.
  - Each iteration produces an **increment**.
- Each iteration comprises **activities** from various **disciplines** (e.g. design, implementation).
  - Activities rely on and produce **artifacts**.

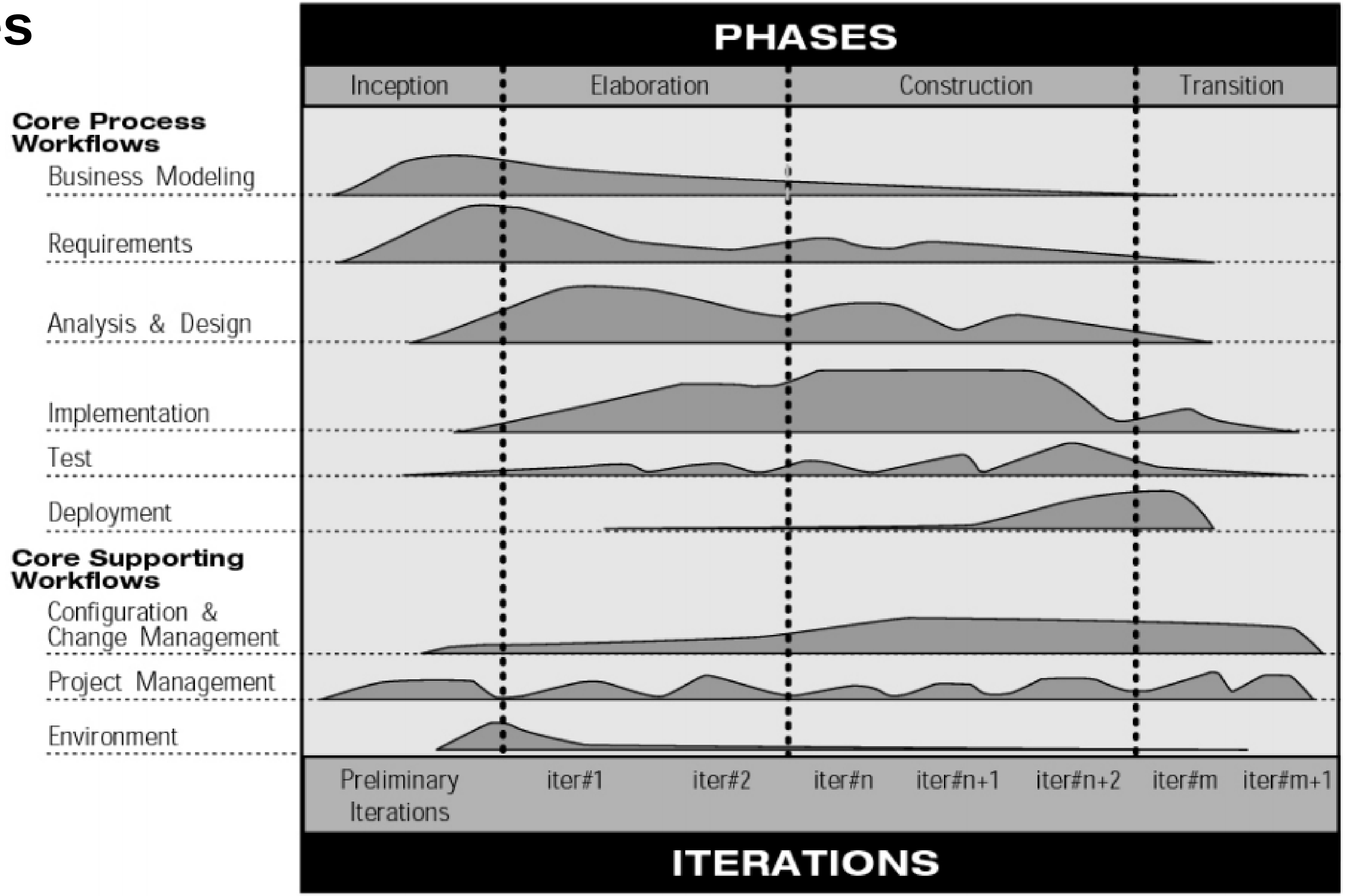
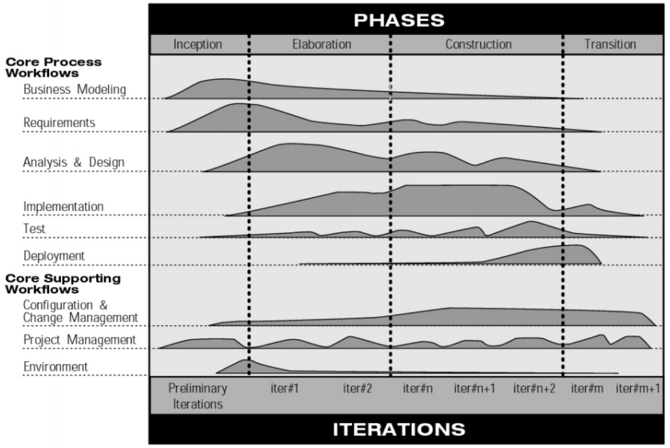


Figure: Rational Software: Rational Unified Process – Best Practices for Software Development Teams. White Paper TP026B, Rev. 11/01

Unified Process Phases



1. **Inception**
   - Approximate vision
   - Business case
   - Scope
   - Vague estimates

Not a “requirements phase”, but feasibility analysis after which we can decide whether it makes sense to proceed with project

2. **Elaboration**
   - Refined vision
   - Identification of most requirements and scope
   - More realistic estimates
   - Iterative implementation of core architecture
   - Resolution of high risks

Not a “design phase”, but iterative development of core architecture and mitigation of high risks

3. **Construction**
   - Iterative implementation of lower-risk and other elements
   - Preparation for deployment

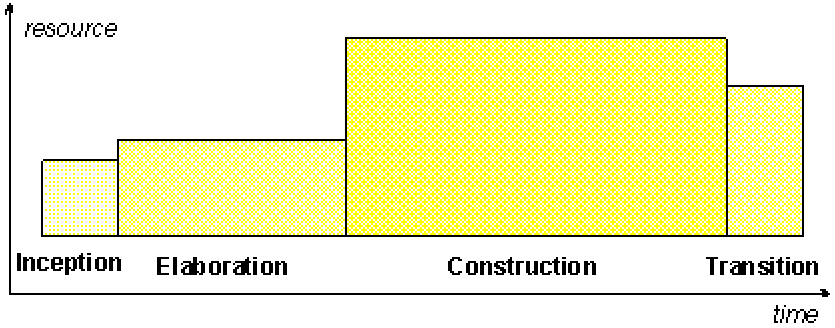
Not an “implementation phase”, but incremental design and development of sets of features

4. **Transition**
   - Beta tests
   - Deployment

Unified Process Phases

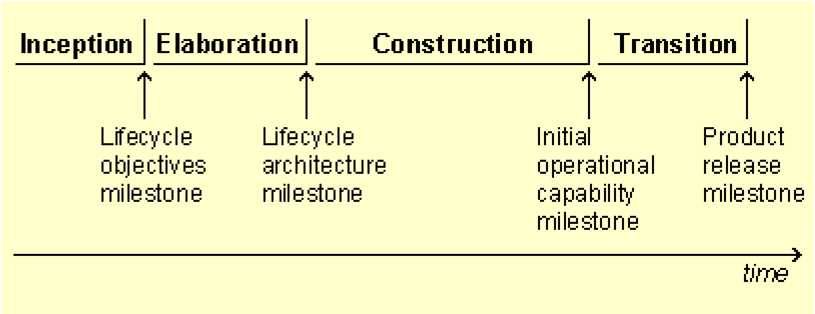
Time and Effort Distribution

- Inception and Transition typically take up to 10% each of time and effort
- Elaboration typically takes at most one third of time and effort
- Construction typically takes at least half of time and effort



Milestones

- Activities within phases overlap
- However, phases are separated by milestones to judge project feasibility and progress
- Milestones have defined deliverables and evaluation criteria



Figures © 1987-2001 Rational Software Corporation

In-Class Quiz #1: Unified Process Phases

1. **Inception**
   - Approximate vision
   - Scope
   - …
2. **Elaboration**
   - More realistic estimates
   - …
3. **Construction**
   - …
   - Preparation for deployment
4. **Transition**
   - …
   - Deployment

- **Which phase do these activities and artifacts belong to?**
  a) Beta tests
  b) Business case
  c) Identification of most requirements
  d) Iterative implementation of core architecture
  e) Iterative implementation of lower-risk and other elements
  f) Refined vision
  g) Resolution of high risks
  h) Vague estimates

Inception: Primary Objectives

- Establish the project’s **software scope and boundary conditions**, including
  - an operational vision
  - acceptance criteria
  - what is intended to be in the product and what is not
- Identify the **critical use cases** of the system
  - i.e. the primary scenarios of operation that will drive the major design trade-offs
- Identify (and maybe demonstrate) at least one **candidate architecture**
  - against the background of some of the primary scenarios
- Estimate the **overall cost and schedule** for the entire project
  - and more detailed estimates for the elaboration phase that will immediately follow
- Estimate **potential risks** (i.e. sources of unpredictability / uncertainty)
- Prepare the **supporting environment** for the project

© 1987-2001 Rational Software Corporation

Inception: Essential Activities

- **Formulating the scope of the project.**
  - This involves capturing the context and the most important requirements and constraints to such an extent that you can derive acceptance criteria for the end product.
- **Planning and preparing a business case.**
  - Evaluating alternatives for risk management, staffing, project plan, and cost/schedule/profitability trade-offs.
- **Synthesizing a candidate architecture,**
  - evaluating trade-offs in design, and in make/buy/reuse, so that cost, schedule and resources can be estimated. The aim here is to demonstrate feasibility through some kind of proof of concept. This may take the form of a model which simulates what is required, or an initial prototype which explores what are considered to be the areas of high risk. *The prototyping effort during inception should be limited to gaining confidence that a solution is possible – the solution is realized during elaboration and construction.*
- **Preparing the environment for the project,**
  - assessing the project and the organization, selecting tools, deciding which parts of the process to improve.

© 1987-2001 Rational Software Corporation

Inception: "Lifecycle Objectives" Milestone

"Ensure that **it makes sense** to do this."

Evaluation Criteria

- Stakeholder concurrence on scope definition and cost/schedule estimates
- Agreement that the right set of requirements have been captured and that there is a shared understanding of these requirements.
- Agreement that the cost/schedule estimates, priorities, risks, and development process are appropriate.
- All risks have been identified and a mitigation strategy exists for each.

Artifacts

- Vision
- Business Case
- Risk List
- Software Development Plan
- Iteration Plan
- Development Case
- Tools
- Glossary
- Use Case Model
- Project Repository
- Use Case Modeling Guidelines
- Domain Model
- Project-Specific Templates
- Prototypes

© 1987-2001 Rational Software Corporation

Elaboration: Primary Objectives

- Establish **baseline architecture** addressing the architecturally significant scenarios
  - which typically expose the top technical risks of the project
- Address all architecturally significant **risks** of the project
- Produce an **executable architecture** and possibly exploratory, throw-away prototypes to mitigate specific risks such as
  - design/requirements trade-offs
  - component reuse
  - product feasibility
  - acceptance of investors, customers, and end-users
- Demonstrate that the baseline architecture will support the **requirements** of the system
  - at a reasonable cost and in a reasonable time
- Establish the **supporting environment** for the project

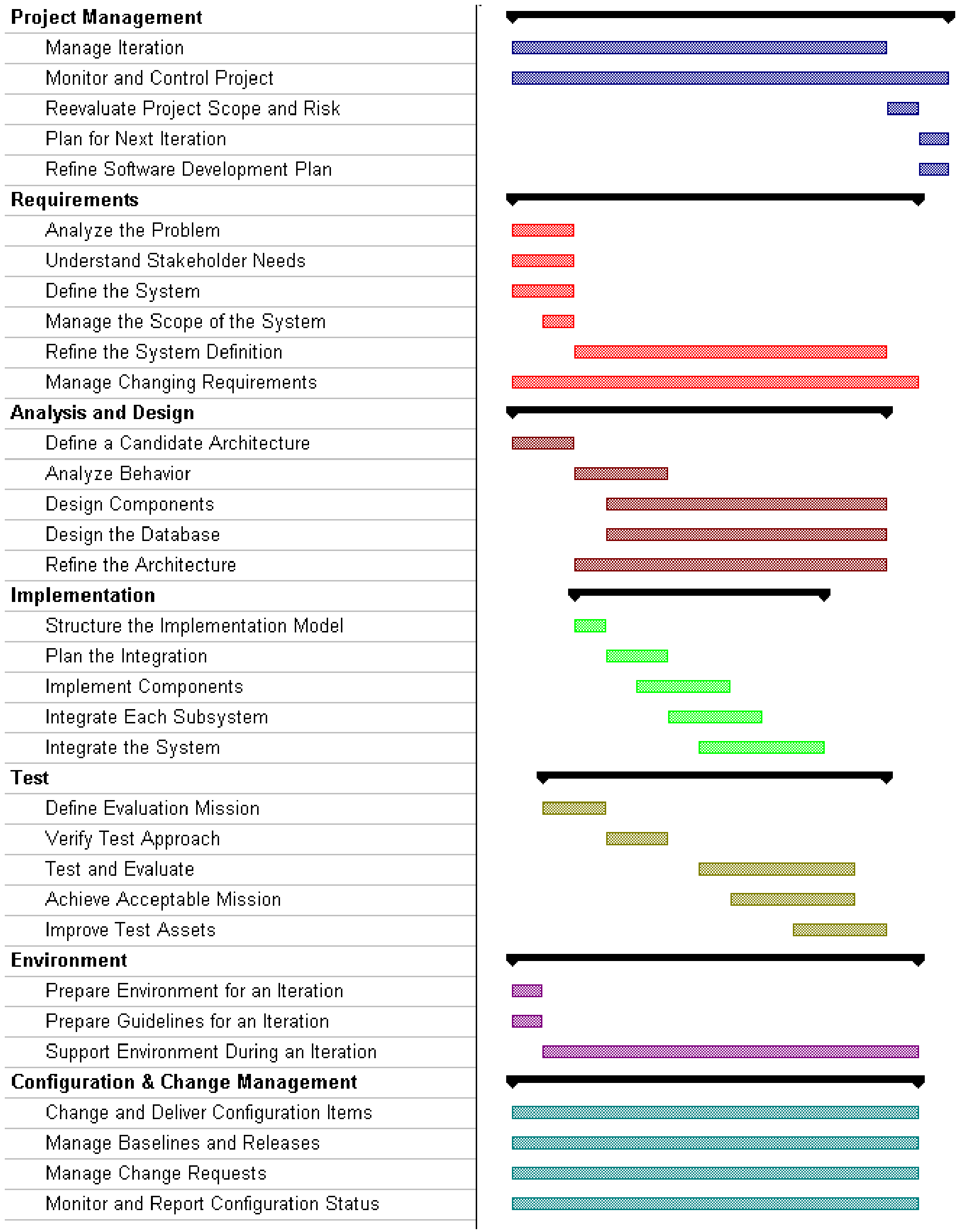
➢ Ensure that…
  - the architecture, requirements and plans are stable enough
  - the risks sufficiently mitigated

…to predictably determine the **cost and schedule** for completion of development

© 1987-2001 Rational Software Corporation

Elaboration: Essential Activities

- **Defining, validating and baselining the architecture.**
- **Refining the vision**
  - based on new information obtained during the phase, establishing a solid understanding of the most critical use cases that drive the architectural and planning decisions.
- **Creating and baselining detailed iteration plans**
  - for the construction phase.
- **Refining the development case and putting the development environment in place,**
  - including the process, tools and automation support required to support the construction team.
- **Refining the architecture and selecting components.**
  - Potential components are evaluated and the make/buy/reuse decisions sufficiently understood to determine the construction phase cost and schedule with confidence.
  - The selected architectural components are integrated and assessed against the primary scenarios.
  - Lessons learned from these activities may well result in a redesign of the architecture, taking into consideration alternative designs or reconsideration of the requirements.



© 1987-2001 Rational Software Corporation

Elaboration: “Lifecycle Architecture” Milestone

“Ensure that **you are really able** to do it.”

Evaluation Criteria

- Product vision and requirements are stable.
- Architecture is stable.
- Key test and evaluation approaches are proven.
- Test and evaluation of executable prototypes have demonstrated that major risk elements have been addressed and credibly resolved.
- Iteration plans for Construction phase are of sufficient detail and fidelity to allow the work to proceed.
- Iteration plans for Construction phase are supported by credible estimates.
- All stakeholders agree that the current vision can be met if the current plan is executed to develop the complete system, in the context of the current architecture.
- Actual resource expenditure versus planned expenditure are acceptable.

Artifacts

- Prototypes
- Risk List
- Development Case
- Tools
- Software Architecture Document
- Design Model
- Data Model
- Implementation Model
- Vision
- Software Development Plan
- Design and Programming Guidelines
- Iteration Plan
- Use Case Model
- Supplementary Specifications
- Test Suite
- Test Automation Architecture
- Business Case
- Analysis Model
- Training Materials
- Project-Specific Templates

© 1987-2001 Rational Software Corporation

An Important Transition

- **During Inception and Elaboration,** the project still is a relatively light-and-fast, low-risk operation.
  - If the project fails to reach one of these milestones,
    - it may be aborted due to infeasibility
    - or it needs to be considerably re-thought.

- **During Construction and Transition,** the project becomes a high-cost, high-risk operation with substantial organizational inertia.
  - If the project fails to reach one of these milestones,
    - deployment may have to be postponed by at least one release.

- Note: The UP is not the *cause* for the high complexity. Rather, it provides the *structure* that makes such highly complex projects manageable at all.

© 1987-2001 Rational Software Corporation