.

.

Getting Started

This content provides guidance on how to get started with this configuration. This area provides information useful for understanding how to deploy and adopt OpenUP.

1.1 Understanding OpenUP

- **1.1.1 Roadmap: OpenUP Roadmap**
- **1.1.2 Who Should Use OpenUP**
- **1.1.3 Core Principles**
- **1.1.4 Minimal, Complete, and Extensible**

1.2 #concept# Basic Process Concepts
1.3 #concept# Practice

.

.

1.1 Understanding OpenUP

OpenUP is a lean Unified Process that applies iterative and incremental approaches within a structured lifecycle. OpenUP embraces a pragmatic, agile philosophy that focuses on the collaborative nature of software development. It is a tools-agnostic, low-ceremony process that can be extended to address a broad variety of project types.

.

.

1.1.1 Roadmap: OpenUP Roadmap

This roadmap presents an overview of OpenUP, its purpose, and its lifecycle.

Overview of OpenUP

OpenUP is for small teams who work together in the same location. The team needs to engage in plenty of daily face-to-face interaction. Team members include the stakeholders, developers, architects, project manager, and testers. They make their own decisions about what they need to work on, what the priorities are, and how best to address stakeholder needs. The organization must support the team by allowing them these responsibilities.

Team members collaborate extensively. The presence of **stakeholders as team members** is critical to successfully implementing OpenUP.

Team members participate in daily stand-up meetings to communicate status and issues. Problems are addressed outside of the daily meetings.

OpenUP focuses on significantly reducing risk early in the lifecycle. This requires regular risk review meetings and rigorous implementation of mitigation strategies.

All work is listed, tracked, and assigned through the **Work Items List**. Team members use this single repository for all tasks that need to be recorded and tracked. This includes all change requests, bugs, and stakeholder requests.

Use cases are used to elicit and describe requirements; therefore, team members need to develop skills in writing good use cases. stakeholders are responsible for reviewing and certifying that the requirements are correct. Use cases are developed collaboratively.

Architecturally significant requirements must be identified and stabilized in the Elaboration phase so that a robust architecture can be created as the core of the system. An architecturally significant requirement change that must be addressed may arise later in development, but the risk of this happening is significantly reduced in the Elaboration iteration.

Testing is performed multiple times per iteration, every time the solution is incremented with the development of a requirement, change, or bug fix. These tests happen after developers have developed the solution (which should have been unit-tested) and integrate it into the code base. These tests help guarantee that a stable build is created and always available as development progresses.

OpenUP does not include content for **deployment**, **configuration management**, or the **development environment** (such as customizing this process or setting up development environments). OpenUP focuses on a single team, and these areas are generally addressed at an organizational or enterprise level. Look for extensions to OpenUP that address these wider areas.

OpenUP is an iterative software development process that is minimal, complete, and extensible. It is governed by four core principles:

- **Collaborate to align interests and share understanding**
- **Balance competing priorities to maximize stakeholder value**
- **Focus on the architecture early to minimize risks and organize development**
- **Evolve to continuously obtain feedback and improve**

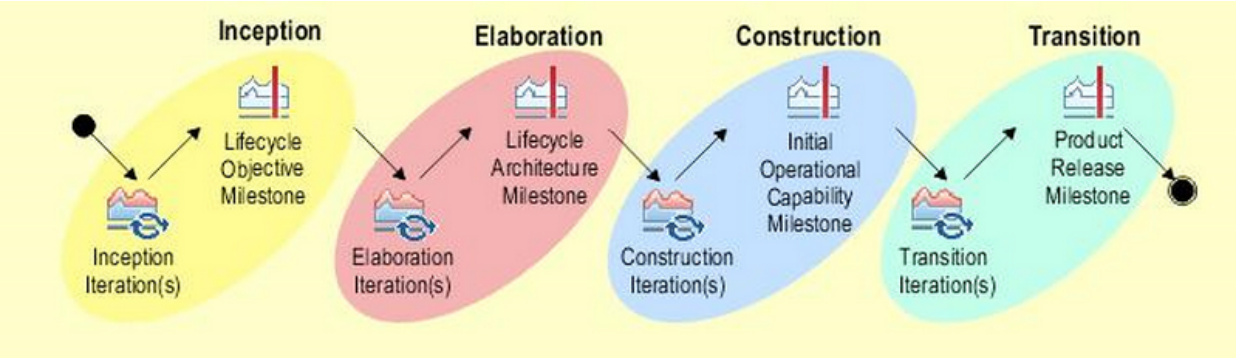
People in roles perform tasks that use and produce artifacts. OpenUP describes the minimal set of **roles**, **tasks (organized by disciplines)**, and **artifacts (organized by work product domains)** involved in software development:

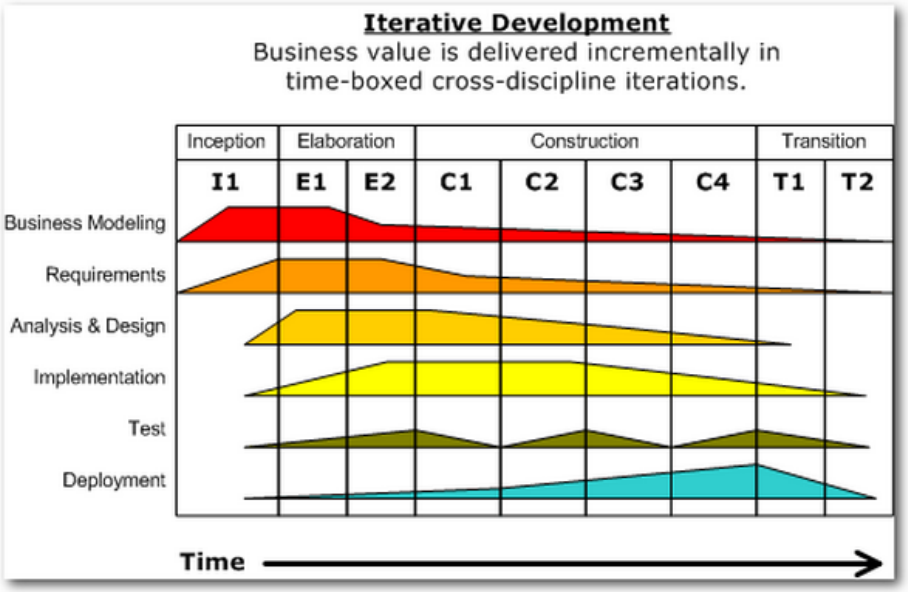
- Roles
- Tasks
- Work Products

The OpenUP software development lifecycle

OpenUP is an iterative process distributed throughout four phases: **Inception**, **Elaboration**, **Construction**, and **Transition**. Each phase consists of one or more iterations where stable, working versions of the software are developed and released, with the completion of each iteration representing a minor milestone for the project and contributing to the successful achievement of the phase's major milestone, where phase objectives are met.

The following diagram depicts OpenUP Lifecycle.





source: wiki

How to get started?

The fourth OpenUP core principle, "Evolve to continuously obtain feedback and improve", suggests an iterative and incremental approach to adopting OpenUP.

- Start with the core principles and understand the intentions behind OpenUP. Then assess your existing process, and select one or two key areas that you would like to improve.
- Begin using OpenUP to improve these areas first.
- In later iterations or development cycles, make incremental improvements in other areas.
- If you have little or no experience with unified processes or other iterative processes, use OpenUP in a small pilot project, perhaps with only three to four people working for only two to three months.

While OpenUP is a ready to use as-is, you may choose to extend the process or modify the work product templates to suit your specific needs. For example:

- You may require more or less precision in your work products.
- Your organization may have specific configuration management practices or safety protocols to include in your process.
- You may simply want to put your own corporate logo on the banner.
- You may want to incorporate lessons learned from a retrospective review into OpenUP.

Use EPF Composer to extend and tailor OpenUP. For more information about EPF composer, visit www.eclipse.org/epf.
.

.
1.1.2 Who Should Use OpenUP
This supporting material explains who should use OpenUP.

OpenUP is most useful for four primary groups of users:

- Software development practitioners (developers, project managers, analysts, and testers) working together as a project team
- stakeholders
- Software process engineers
- Instructors

Software development practitioners can find guidance on what is required of them in the roles defined by OpenUP. Each role describes a set of activities and artifacts for which the role is responsible. Guidance is also given on how those roles collaborate.

Stakeholders will find guidance on what they may expect from the software development team, and how the software will be created. OpenUP also describes the stakeholders' responsibilities, and states how they can best work with the development team to obtain software that meets their needs.

Software process engineers can use EPF Composer to extend and modify OpenUP. Modification may be as simple as altering templates for work products, or as sophisticated as adding activities necessary for creating software in your specific environment (such as audits for safety-critical systems). In addition to modifying method content, process engineers can add, change, or remove process flows to add organization-specific capability patterns.

OpenUP is appropriate for academic organizations also. As an open source process, it can serve as the basis for software engineering courses and, when combined with the EPF Composer, courses in software process engineering.

.

.
1.1.3 Core Principles
Four core principles capture the general intentions behind OpenUP. They create the foundation for interpreting roles and work products, and for performing tasks.

Balance competing priorities to maximize stakeholder value
Promote practices that allow project participants and stakeholders to develop a solution that maximizes stakeholder benefits, and is compliant with constraints placed on the project.

See **#concept# Balance core principle for more information.**

Collaborate to align interests and share understanding
Promote practices that foster a healthy team environment, enable collaboration and develop a shared understanding of the project.

See **#concept# Collaborate core principle for more information.**

Focus on the architecture early to minimize risks and organize development
Promote practices that allow the team to focus on architecture to minimize risks and organize development.

See **#concept# Focus core principle for more information.**

Evolve to continuously obtain feedback and improve
Promote practices that allow the team to get early and continuous feedback from stakeholders, and demonstrate incremental value to them.

See **#concept# Evolve core principle for more information.**
.

.
1.1.4 Minimal, Complete, and Extensible
OpenUP is minimal, complete, and extensible. It's the minimum amount of process for a small team. It can be used as-is or extended and customized for specific purposes.

Introduction
A good way to understand OpenUP is to think of it as targeted to teams that have the following objectives:

- Apply the minimally necessary process that brings value.
- Avoid being overloaded with unproductive formal work products.
- Use a process that can be tailored and extended for additional needs that may arise during the software development lifecycle.

In order to be applied to as many project environments as possible, OpenUP is a process that has the following characteristics:

- Minimal - only fundamental content is included
- Complete - it can be manifested as an entire process to build a system
- Extensible - it can be used as a foundation on which process content can be added or tailored as needed

Minimal
OpenUP is considered an agile, lightweight process that promotes software development best practices such as:

- iterative development
- team collaboration
- continuous integration and tests
- frequent deliveries of working software
- adaptation to changes, and so on.

Other factors also count when determining the size of process material, such as the number of method elements like roles, tasks, artifacts and guidance.

OpenUP provides descriptions of responsibilities, skills and competencies of the fundamental roles in a team. See the list of **04.Roles** for more details.

OpenUP provides the essential artifacts needed to capture and communicate decisions. Ultimately, the process is not governed by artifact creation. For

example, thinking about design is more important than documenting the design; assessing and planning an iteration is about promoting team collaboration instead of creating plans that are written in stone. In addition, for each artifact, OpenUP suggests informal representations or provides informal templates as starting points for teams that do not need to create their own. The team decides what the most appropriate form to be applied is. See the list of **05.Work Products** organized by domains.

Tasks in OpenUP are clear and focused on results. Text is usually short and objective, describing how individuals should collaborate to achieve objectives. Steps are short descriptions of what to achieve, and point to external guidance on how to do it. See the list of **06.Tasks** organized by disciplines.

OpenUP recommends the least amount of process guidance a team should use in order to be successful. Teams may have different names or responsibilities for roles, may perform tasks in or have different representations for artifacts, but they still want to follow the principles and practices described in OpenUP to increase the chances of project success.

Complete
OpenUP is considered a complete process because it covers the essential disciplines in a software development lifecycle by guiding the team in the following high level activities:

- Customer needs and requirements are elicited and captured, with continuous customer involvement.
- A robust architecture for the system is evolved, addressing technical risks and promoting team organization.
- For each requirement, a technical solution is designed, implemented and tested, which conforms to the architectural decisions.
- The system is evaluated by tests that validate customer requirements.
- Defects and enhancements feed back into development.
- Work is prioritized, iterations are planned and assessed, and team members take on work to be done.

However, OpenUP assumes that the project team is not responsible for certain activities and decisions that are assigned to other areas of the organization, such as:

- Creation of the **business case** is dealt with by management who decides whether or not the project is worth investing in, what is the return on investment, and so on.
- **Environment setup** - some organizational issues may not be in the scope of the team, such as: installation, configuration and licensing of development tools and configuration management tools; development process customization and publishing, etc.
- **Deployment and operation** - addressed at an organizational rather than a team level.

Other areas of concern are also not present in OpenUP, because small teams do not need to deal with the formality or overhead of these areas. They include, but are not limited to:

- Advanced configuration management
- Advanced requirements management
- Program and portfolio management

Extensible
OpenUP can be adopted by projects as it is, out-of-the-box, because it's minimal and complete.
However, different projects may have different needs. Organizations may be interested in customizing some aspects of OpenUP to adapt it to their projects. These are some examples of possible customization:

- Add new or rename existing roles
- Add steps to existing tasks
- Create new guidelines on a given technique
- Remove a given content area
- Modify existing or add new templates to artifacts
- Modify or create a new process lifecycle
- Add or remove process content
- Add guidance on how to achieve compliance
- Add guidance specific to a technology being used
- Replace or augment one of the layers with new content (e.g. by changing or adding material to how management is performed)
- Etc.

OpenUP can be customized and extended by using the EPF Composer tool, which allows you to author, configure and publish methods. With EPF Composer, you can add, remove and change elements according to your team needs, then publish the content to serve as guidance for your team.

See more information about customization in **#guideline# Resources for Customizing Methods.**
.

.
1.2 #Concept# Basic Process Concepts

The Basic Elements
The basic elements of a process website are:

- **Work product:** what is produced
- **Task:** how to perform the work
- **Role:** who performs the work
- **Process:** used to define work breakdown and workflow
- **Guidance:** templates, checklists, examples, guidelines, concepts, and so on.

These "basic elements" are the building blocks from which processes are composed.

Organizing Elements
The basic elements are organized using the following elements.

- **Practice**

A practice is a documented approach to solving one or several commonly occurring problems. Practices are intended as "chunks" of process for adoption, enablement, and configuration. Practices are built from the basic elements described above.

- **Configuration**

From the end-user perspective, a configuration is a selection of method content to be published. Most configurations consist of a selection of practices plus some content to tie the practices together. The published configuration is often loosely referred to as a process website.

Details and Examples
The following provides more detail about the basic elements and provides some examples.

- **Work product**

Work products may take various shapes or forms, such as:

  - Documents, such as a Vision, or a Project Plan.
  - A model, such as a Use-Case Model or a Design Model. These can contain model elements (sub-artifacts) such as Design Classes, Use Cases, and Design Subsystems.
  - Databases, spreadsheets, and other information repositories.
  - Source code and executables.

Work products can be classified as "**artifacts**" if they are concrete things, "**outcomes**" if they are not concrete, and "**deliverables**" if they are a packaging of artifacts.

- **Role**

A role defines the behavior and responsibilities of an individual, or a set of individuals working together as a team, within the context of a software engineering organization. Note that roles are not individuals; instead, roles describe responsibilities. An individual will typically take on several roles at one time, and frequently will change roles over the duration of the project.

Some examples:

  - Analyst - Represents customers and end users, gathers input from stakeholders and defines requirements.
  - Developer - Develops a part of the system, including designing, implementing, unit testing, and integrating.

- **Task**

A task is work performed by a role. It is usually defined as a series of steps that involve creating or updating one or more work products. Some examples:

  - Develop a vision - Develop an overall vision for the system, including capturing the problem to be solved, the key stakeholders, the scope and boundary of the system, the system's key features, and any constraints.
  - Plan Iteration - Define the scope and responsibilities of a single iteration.

- **Process**

Processes pull together tasks, work products, and roles, and add structure and sequencing information. Tasks or work products can be grouped into higher level activities, called a work breakdown structure (WBS). Activities or tasks can be marked as "planned" to identify work that you expect to assign and track.

Iteration, -> **Activity**, -> **Task**, -> **Step**,

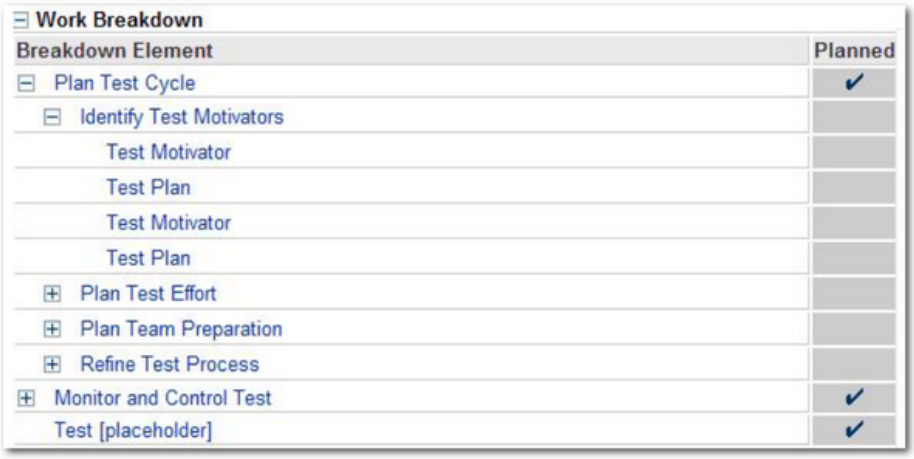


Figure 1: Example Work Breakdown

Diagrams can be added to providing sequencing information. The following example shows an initial **activity**, "Plan Test Cycle", followed by two **activities** that go in parallel, "Monitor and Control Test" and "Test".

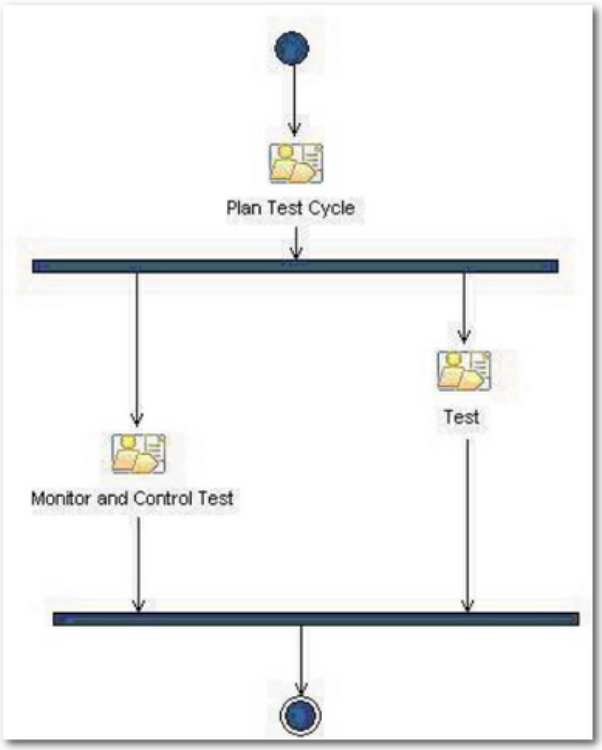


Figure 2: Example **Activity** Diagram

Note that a reusable partial process is sometimes referred to as a capability pattern.

For More Information

More in-depth material on these concepts is generally found in articles on EPF Composer and Rational(R) Method Composer, which use these concepts as building blocks.

- To learn more about the Eclipse Process Framework Project and EPF Composer, visit http://www.eclipse.org/epf and http://www-128.ibm.com/developerworks/rational/library/05/1011_kroll/index.html
- For more information on Rational Method Composer, see:
  - The RMC Product Page on developer works http://www-306.ibm.com/software/awdtools/rmc/
  - Article: "IBM Rational Method Composer: Part 1: Key concepts" at http://www.ibm.com/developerworks/rational/library/dec05/haumer/index.html
- For an in-depth explanation of the meta-model on which EPF Composer and Rational Method Composer are based, see:
  - OMG, "Software Process Engineering Meta model," version 1.1, formal/2005-01-06, 2005. http://www.omg.org/technology/documents/formal/spem.htm

.

.

1.3 #Concept# Practice

A practice is an approach to solving one or several commonly occurring problems. Practices are intended as "chunks" of process for adoption, enablement, and configuration.

Why Practices?

Practices enable a compositional approach to building methods. This approach offers the following benefits:

- Adaptability and scalability
- Incremental adoption
- Easy to configure and use
- Community development

Adaptability and Scalability

Practices can be adapted to support a range of solutions. In particular, practices can be adapted to suit your organization and supplemented by your own practices.

The core practices are based on a common framework that allows them to be composed. These core practices are tool-agnostic, low-ceremony practices that can be extended to address a broad variety of development concerns, such as SOA, geographical distribution, model-driven architecture and embedded systems. Tool and technology specific guidance can be added, such as guidance on J2EE, and a variety of development tools. Some of these extensions can be quite modest, adding for example just tool specific guidance to existing tasks, while others can be comprehensive, defining processes that provide a radically expanded scope with new or altered artifacts, new or altered tasks, and new or altered roles.

Extensions and additions to the practices can be the following:

- being used internally by an organization;
- open source as a part of the Eclipse Process Framework (EPF) project;
- made freely available outside the open source licenses of Eclipse (EPL);
- sold commercially as an extension to the basic framework, such as the IBM(R) Practices.

Incremental Adoption
Each practice is described as a standalone capability that can be adopted by an organization or project. Each practice may include enablement materials that explain how to get started.

Easy to Configure and Use
Creating a method is as simple as selecting the practices that you wish to adopt, and then publishing the results. Each practice adds itself into the framework so that content can be viewed by practice, or across practices by work product, role, task and so on.

Community Development
Since a practice can be easily authored on its own, practices are ideal for community development. The basic agile practices for the EPF Practices are, in fact, developed by the Eclipse Process Framework community.

.

.