

Unified Process – The Inception Phase

Marek Rychlý

`rychly@fit.vutbr.cz`

Brno University of Technology
Faculty of Information Technology
Department of Information Systems

Information Systems Analysis and Design (AIS)
24 October 2019



- 1 Unified Process
 - History
 - Characteristics
 - Phases and Milestones

- 2 The Inception Phase
 - Requirements Analysis, FURPS+
 - Use Cases
 - Requirement Elicitation Techniques

History of Unified Process

- In 1987, Jacobson started Objective Systems/Objectory AB.
 - invented use cases to specify functional SW requirements
 - developed Object-Oriented Software Engineering/OOSE method (a light version of the commercial SW process Objectory/Object Factory)
- In 1995, Rational acquired Objectory and Jacobson&Booch&Rumbaugh
 - designed the UML in 1996 that was standardised in 1997 by OMG (from OOSE, Booch method, Rumbaugh object-modelling technique/OMT)
 - improved Objectory Process to Rational Unified Process in 1998 (under the leadership of Philippe Kruchten from Rational)
- In 1999, Ivar Jacobson, Grady Booch and James Rumbaugh published book “The Unified Software Development Process”.
- In 2003, IBM acquired Rational Software and OpenUP method in 2006. (an open-source subset of RUP tailored for the delivery of Agile projects)
- In 2012, Ambler&Lines published book “Disciplined Agile Delivery”/DAD. (it adopts and extends strategies from UP, Scrum, XP, and others)

Unified Process Characteristics

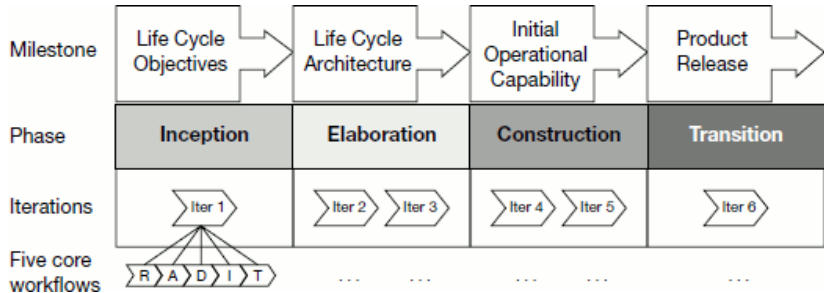
- A generic SW development process that has to be adapted.
(by a software development organisation before its application in projects)
- Policies, templates, tools, database, SW life-cycle aspects, etc.
- Three characteristics of UP:
 - risk-focused
(addressing the most critical risks early in the project life cycle)
 - architecture-centric
(multiple architectural models and views for a robust architecture)
 - iterative and incremental
(phases are divided into a series of time-boxed iterations, each of them results in an increment, a new baseline/release of a software system)
- Each iteration is a complete SW development sub-process.
(planning, specification and requirements analysis, design and implementation, integration and testing, internal/public release, evaluation, etc.)



(Assessment, Planning, and Project Specification are not included in UP)



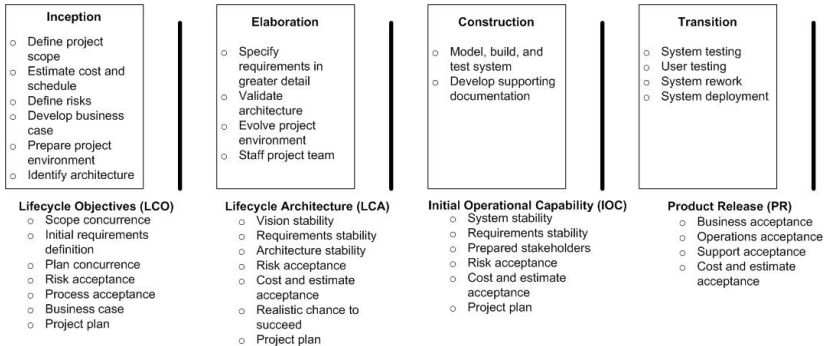
Unified Process Structure



(adopted from "UML 2 and the Unified Process" by Arlow&Neustadt)

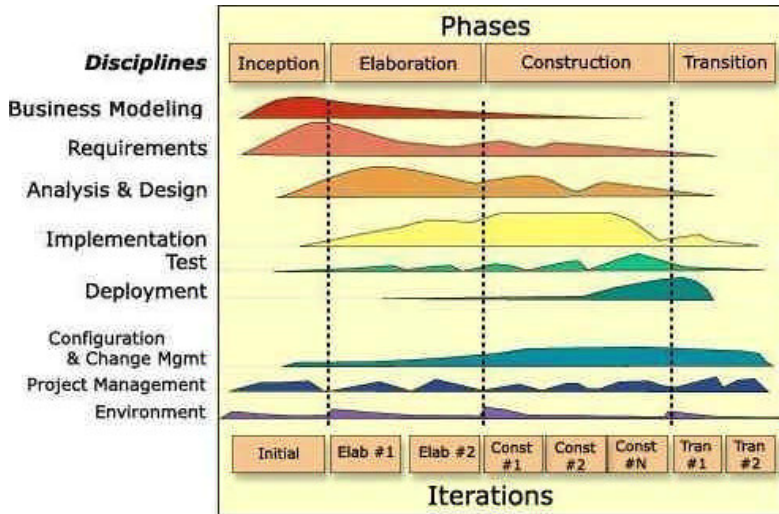
- A mid-size software project takes about 18 months.
- For each phase, there are defined goals and a focus on one or more particular software development tasks.

(Rational) Unified Process Milestones



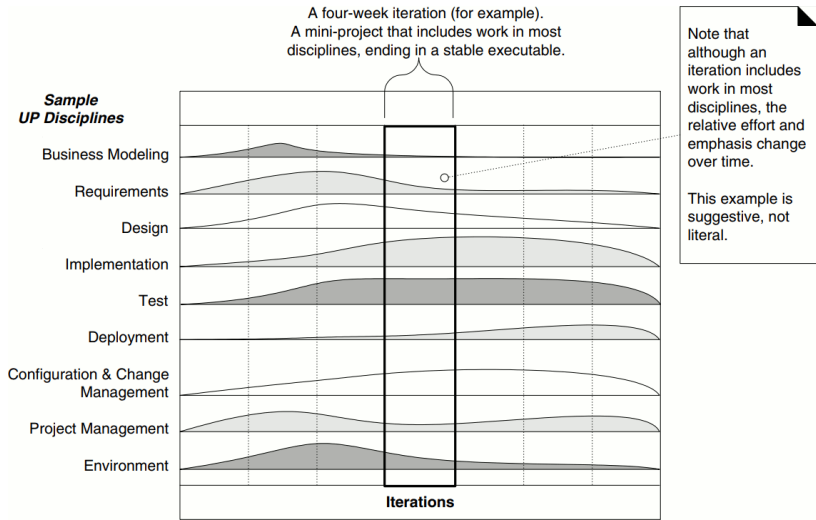
(adopted from “A Manager’s Introduction to The Rational Unified Process” by Scott W. Ambler)

(Rational) Unified Process Phases



(adopted from "A Manager's Introduction to The Rational Unified Process" by Scott W. Ambler)

Unified Process Disciplines through Iterations




(adopted from "Applying UML and Patterns" by Craig Larman)

The Inception Phase

Goals to determine the system's **life-cycle objectives**:

- establishing feasibility,
(feasibility analysis of technical decisions and business requirements, e.g., by building a technical, or proof of concept, prototype)
- creating a business case,
(so the project's business benefit is quantifiable)
- capturing essential requirements,
(to define a scope of the software system)
- identifying critical risks.

Focus:

- requirements,
- analysis,
- design and implementation (if it was decided to build a prototype). 

Goals/Milestones of the Inception Phase

Conditions of satisfaction	Deliverable
The stakeholders have agreed on the project objectives	A vision document that states the project's main requirements, features, and constraints
System scope has been defined and agreed on with the stakeholders	An initial use case model (only about 10% to 20% complete)
Key requirements have been captured and agreed on with the stakeholders	A project glossary
Cost and schedule estimates have been agreed on with the stakeholders	An initial project plan
A business case has been raised by the project manager	A business case
The project manager has performed a risk assessment	A risk assessment document or database
Feasibility has been confirmed through technical studies and/or prototyping	One or more throwaway prototypes
An architecture has been outlined	An initial architecture document

(adopted from "UML 2 and the Unified Process" by Arlow&Neustadt)

The Elaboration Phase

Goals to determine the system's **life-cycle architecture**:

- create and executable architectural baseline for the system,
- refine the risk assessment,
- define the system's quality attributes,
- capture 80% of the functional requirements,
- create a detailed construction phase plan,
- estimate resources, time, equipment, staff, and cost.

Focus:

- requirements – to specify the requirements and scope,
- analysis – to defined what will be constructed,
- design – to create a stable system architecture,
- implementation – to build the architectural baseline,
- testing – to test the architectural baseline.



Goals/Milestones of the Elaboration Phase

Conditions of satisfaction	Deliverable
A resilient, robust executable architectural baseline has been created	The executable architectural baseline
The executable architectural baseline demonstrates that important risks have been identified and resolved	UML static model UML dynamic model UML use case model
The vision of the product has stabilized	Vision document
The risk assessment has been revised	Updated risk assessment
The business case has been revised and agreed with the stakeholders	Updated business case
A project plan has been created in sufficient detail to enable a realistic bid to be formulated for time, money, and resources in the next phases	Updated project plan
The stakeholders agree to the project plan	
The business case has been verified against the project plan	Business case
Agreement is reached with the stakeholders to continue the project	Sign-off document

(adopted from "UML 2 and the Unified Process" by Arlow&Neustadt)

The Construction Phase

Goals to build the initial **operational system**:

- complete all requirements, analysis and design,
- evolve architectural baseline into the final system.

Focus:

- requirements – to specify the rest of requirements,
- analysis – to finish the analytical models,
- design – to finish the design models,
- implementation – to build the initial operational system,
- testing – to test the initial operational system.

Goals/Milestones of the Construction Phase

Conditions of satisfaction	Deliverable
The software product is sufficiently stable and of sufficient quality to be deployed in the user community	The software product The UML model Test suite
The stakeholders have agreed and are ready for the transition of the software to their environment	User manuals Description of this release
The actual expenditures vs. the planned expenditures are acceptable	Project plan

(adopted from "UML 2 and the Unified Process" by Arlow&Neustadt)

The Transition Phase

Goals to release the **production version** to the customer:

- fix defects,
- prepare the user site for the new software,
- tailor the software to operate at the user site,
- modify the software if unforeseen problems arise,
- create user manuals and other documentation,
- provide user consultancy,
- conduct a post project review.

Focus:

- requirements – none,
- analysis – none,
- design – to fix design models in the case of beta-testing issues,
- implementation – to tailor the software for the user site, to fix bugs,
- testing – a beta-testing and acceptance testing at the user site.



Goals/Milestones of the Transition Phase

Conditions of satisfaction	Deliverable
Beta testing is completed, necessary changes have been made, and the users agree that the system has been successfully deployed	The software product
The user community is actively using the product	
Product support strategies have been agreed on with the users and implemented	User support plan Updated user manuals

(adopted from "UML 2 and the Unified Process" by Arlow&Neustadt)

The State at the End of Each Phase

Phase	Project State	Outcome
Inception	The stakeholders are discussing the value and feasibility of the solution. The approach to be taken is been agreed.	<i>Project viability agreed.</i> <i>Agreement to fund the Project (Elaboration at least).</i>
Elaboration	Demonstrable, executable versions of the system are being produced to actively mitigate the most serious project risks and prove the approach selected.	<i>Selected approach proven.</i> <i>A stable, proven, executable architecture.</i>
Construction	Deployable solutions, that work end-to-end, are being regularly produced.	<i>Useable solution available</i> <i>A useful, tested, deployable, and documented solution</i>
Transition	The new system is being supported in the live environment. Feedback is being generated from actual system use.	<i>Project complete.</i> <i>The solution is in 'actual' use.</i>

(adopted from "Iterative Project Management: Controlling an Iterative Project" by Ivar Jacobson)

- The phases are not time-boxed, contrary to the iterations.
(a phase is not over until its milestone is achieved)
- A project stays in a particular phase until its milestone is accommodated.
(regardless of what the project's schedule says etc.)

An Adaptation of UP to Agile

Discipline	Practice	Artifact Iteration→	Incep. I1	Elab. E1..En	Const. C1..Cn	Trans. T1..T2
Business Modeling	agile modeling req. workshop	Domain Model		s		
Requirements	req. workshop vision box exercise dot voting	Use-Case Model	s	r		
		Vision	s	r		
		Supplementary Specification	s	r		
		Glossary	s	r		
Design	agile modeling test-driven dev.	Design Model		s	r	
		SW Architecture Document		s		
		Data Model		s	r	
Implementa- tion	test-driven dev. pair programming continuous integration coding standards	...				
Project Management	agile PM daily Scrum meeting	...				
...						

Table 2.1 Sample Development Case. s - start; r - refine
(adopted from "Applying UML and Patterns" by Craig Larman)

The Inception Phase

- An initial short step to establish a common vision and basic scope.
(it is necessary for the most of software projects)
- It should be short – one or a few weeks, it may even be shorter.
(about deciding if the project is worth a serious investigation during elaboration, not to do that investigation; usually without UML diagrams)
- To answer the following without full detailed requirements.
(e.g., perhaps only 10% of the use cases and critical non-functional requirements)
 - What is the vision and scope of a project?
(create a business case or vision artefact, analyse value, etc.)
 - Feasibility?
(an identification of main risks; being honest and realistic is not always easy)
 - Buy components and glue or build from scratch?
 - What is a very rough cost and schedule estimate?
(plans and estimates created during this phase are not reliable)
 - Should we stop or continue with the project?
(feasible from a technical- and useful from a business point-of-view)



Sample Artefacts of the Inception Phase

Artifact [†]	Comment
Vision and Business Case	Describes the high-level goals and constraints, the business case, and provides an executive summary.
Use-Case Model	Describes the functional requirements. During inception, the names of most use cases will be identified, and perhaps 10% of the use cases will be analyzed in detail.
Supplementary Specification	Describes other requirements, mostly non-functional. During inception, it is useful to have some idea of the key non-functional requirements that have will have a major impact on the architecture.
Glossary	Key domain terminology, and data dictionary.
Risk List & Risk Management Plan	Describes the risks (business, technical, resource, schedule) and ideas for their mitigation or response.
Prototypes and proof-of-concepts	To clarify the vision, and validate technical ideas.
Iteration Plan	Describes what to do in the first elaboration iteration.
Phase Plan & Software Development Plan	Low-precision guess for elaboration phase duration and effort. Tools, people, education, and other resources.
Development Case	A description of the customized UP steps and artifacts for this project. In the UP, one always customizes it for the project.

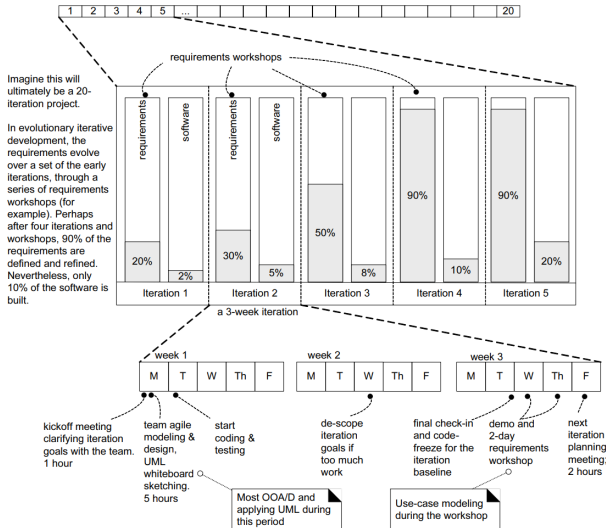
(adopted from "UML 2 and the Unified Process" by Arlow&Neustadt)

Requirements and the Unified Process

- Requirements are considered to be unstable, evolving in time.
(which significantly differs from the point-of-view of the waterfall model)
- Coding and testing typically starts when 10 to 20% of the use cases and critical non-functional requirements are defined.
(the use cases and requirements affecting the system architecture, value, or risks)
- According to Gartner and other studies in 2001/2002,
 - the waterfall was the main issue in 82% of unsuccessful projects
 - and 65% of requirements was defined too early without any value.
- The requirements have to be systematically gathered, documented, organised, and monitored during the time.
- It is useful to go along with the requirement elicitation process.
(gathering, organising, negotiation&discussion, and specification of the requirements with participating end-users)
- It is good to use requirement elicitation techniques.
(interview, surveys, questionnaires, task&domain analysis, brainstorming, prototyping, observation, etc.)



Requirements/Software Progress through Iterations



(adopted from "Applying UML and Patterns" by Craig Larman)

Requirements Analysis with FURPS+ Model

Functionality by a capability (a feature set); re-usability (compatibility, interoperability, and portability); and security (safety and exploit-ability).

Usability by human factors; documentation; consistency/responsiveness; etc.

Reliability by an availability (failure frequency); recoverability (failure time-length); predictability (stability); and accuracy (frequency/severity of errors in outputs).

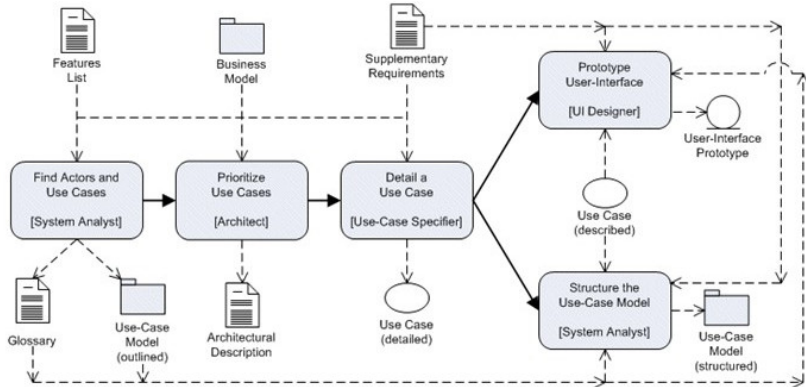
Performance by a throughput; capacity; speed; efficiency; and resource consumption.

Supportability by a serviceability; maintainability; testability; flexibility (modifiability, configurability, adaptability, extensibility, modularity); install-ability; scalability; etc.

FURPS+: System Architecture and Artefacts

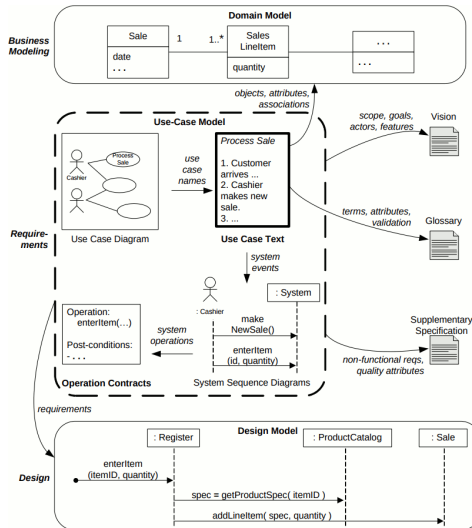
- There are another, previously unclassified, requirements given by
 - implementation (languages, tools, methods, etc.),
 - interfaces (integration of another/legacy systems),
 - operation (in a deployment environment),
 - physical restrictions (storage, topology, etc.),
 - or legal (licences, policies, etc.).
- Also qualitative requirements (i.e., URPS) can significantly affect the architecture and need to be analysed in the inception phase.
- Typical artefacts of the requirement analysis are
 - UML Use Case diagrams,
 - supplementary specification,
(non-functional and out of scope of the use cases)
 - dictionary,
(important key-terms and data dictionary),
 - vision,
(high-level requirements, business cases)
 - business and domain rules (of application domain).

Requirements Workflow



(adopted from "Object-oriented Analysis and Design with the Unified Process" by Jackson&Burd&Sattinger, 2004, ISBN 0619216433)

Sample UP Artefact Relationships



(adopted from "Applying UML and Patterns" by Craig Larman)

Use Case Model

- Use cases are text documents in UP.
(their modelling is mainly about writing such documents)
- Focus on the text specification of use cases, not on diagrams.
(the wrong focus is frequent mistake in this phase)
- End-users must participate in the requirement elicitation process.
- For goals and scenarios important from the user's perspective.
- The text description of the use cases should be
 - short – one paragraph for a typical main successful scenario,
 - informal/common – including alternative scenarios,
 - full – a structured detailed description of steps and variants.
(only for significant scenarios, for the system architecture and end-users)

An Example: Process Sale Use Case (1)

Use Case UC1: Process Sale

Scope: NextGen POS application

Level: user goal

Primary Actor: Cashier

Stakeholders and Interests:

- Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.
- Salesperson: Wants sales commissions updated.
- Customer: Wants purchase and fast service with minimal effort. Wants easily visible display of entered items and prices. Wants proof of purchase to support returns.
- Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.
- Manager: Wants to be able to quickly perform override operations, and easily debug Cashier problems.
- Government Tax Agencies: Want to collect tax from every sale. May be multiple agencies, such as national, state, and county.
- Payment Authorization Service: Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Preconditions: Cashier is identified and authenticated.

Success Guarantee (or Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

(adopted from “Applying UML and Patterns” by Craig Larman)



An Example: Process Sale Use Case (2)

Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Price calculated from a set of price rules.
- Cashier repeats steps 3-4 until indicates done.*
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

- *a. At any time, Manager requests an override operation:
 1. System enters Manager-authorized mode.
 2. Manager or Cashier performs one Manager-mode operation. e.g., cash balance change, resume a suspended sale on another register, void a sale, etc.
 3. System reverts to Cashier-authorized mode.
- *b. At any time, System fails:
 - To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.
 - 1. Cashier restarts System, logs in, and requests recovery of prior state.
 - 2. System reconstructs prior state.
 - 2a. System detects anomalies preventing recovery:
 1. System signals error to the Cashier, records the error, and enters a clean state.
 2. Cashier starts a new sale.

(adopted from "Applying UML and Patterns" by Craig Larman)

An Example: Process Sale Use Case (3)

Special Requirements:

- Touch screen UI on a large flat panel monitor. Text must be visible from 1 meter.
- Credit authorization response within 30 seconds 90% of the time.
- Somehow, we want robust recovery when access to remote services such the inventory system is failing.
- Language internationalization on the text displayed.
- Pluggable business rules to be insertable at steps 3 and 7.
- ...

Technology and Data Variations List:

- *a. Manager override entered by swiping an override card through a card reader, or entering an authorization code via the keyboard.
- 3a. Item identifier entered by bar code laser scanner (if bar code is present) or keyboard.
- 3b. Item identifier may be any UPC, EAN, JAN, or SKU coding scheme.
- 7a. Credit account information entered by card reader or keyboard.
- 7b. Credit payment signature captured on paper receipt. But within two years, we predict many customers will want digital signature capture.

Frequency of Occurrence: Could be nearly continuous.

Open Issues:

- What are the tax law variations?
- Explore the remote service recovery issue.
- What customization is needed for different businesses?
- Must a cashier take their cash drawer when they log out?
- Can the customer directly use the card reader, or does the cashier have to do it?

(adopted from “Applying UML and Patterns” by Craig Larman)



Best-practices for Use Case Modelling

- Focus on goals (essentials), not on particular solution (e.g., UI).
("A user identifies him/herself.", not "A user fills in his/her username&password.")
 - Use brief/informative descriptions, nothing beyond the scenarios.
 - The system is a black-box (describe what it is doing, not how).
("A system registers the order.", not "A system submit SQL INSERT into a DB.")
 - Describe the use cases from point of view of primary users.
(focus on their goals when interacting with the system)
- 1 Define the scope of the system.
 - 2 Identify primary actors and their goals (or vice versa).
 - Who is starting and stopping the system?
 - Who is managing users and their permissions?
 - Can be time an actor and in what use cases?
 - Are there any external systems utilised by or utilising the system?
 - 3 Define and describe the use cases.



Identification of Use Cases

- Start with
 - an actor and his/her user cases (and continue with next one),
 - or with a list of all actors (and continue with their use cases).
- Or identify (external) events, sources/actors and goals/use cases.
(e.g., “enter a new order item” by “cashier” with goal “process item”)
- Create-update-delete (CRUD) use cases should be aggregated.
(e.g., “Manage customers”, not “Insert”/“Update”/“Delete a customer”)
- Some use cases may be questionable (Are they valid?).
 - Negotiate a supplier contract.
 - Handle returns.
 - Log on to the system.
 - Move piece on game board.
- Rather than asking “What is a valid use case?”, we should ask
“What is a useful level of focus to express use cases for application requirements analysis?”



Tests to Find Useful Use Cases

(adopted from “Use Cases” by Peter Kemper)

- **The Boss test** — “What have you been doing all day?”
(you cannot reply “logging on”, a boss would not be happy, no value)
- **The Elementary Business Process (EBP) test**
 (“approve credit” or “price order”, not “delete a line item” or “print the document”)
 - a task performed by one person in one place at one time,
 - in response to a business event,
 - which adds measurable business value and leaves data in a consistent state.
- **The Size test**
Just a single step in a sequence of others → not good!

Applying the Tests to the Questionable Use Cases

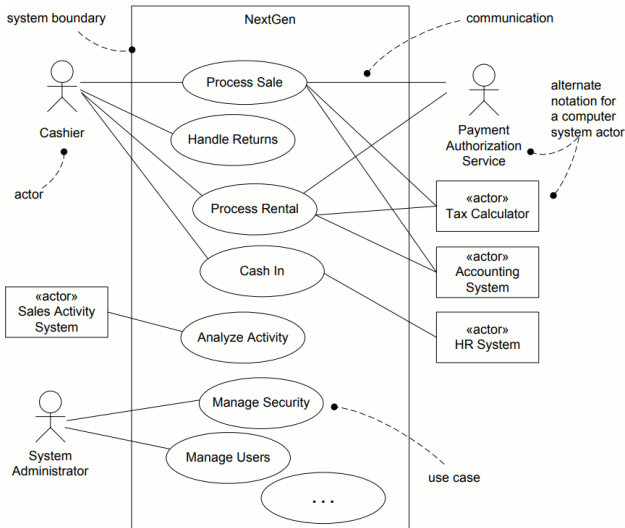
(the examples adopted from “Use Cases” by Peter Kemper)

- **Negotiate a supplier contract?**
(much broader than EBP, rather a business use case)
- **Handle returns?**
(OK with the Boss test, also EBP and Size are good)
- **Log on to the system?**
(Boss is not happy, this is all you do all day)
- **Move piece on game board?**
(it is a single step, so it fails the Size test)

However, there are also acceptable use cases breaking the tests.
(e.g., “Make a credit card payment” is usable in a separate use case)



UML Use Case Diagrams

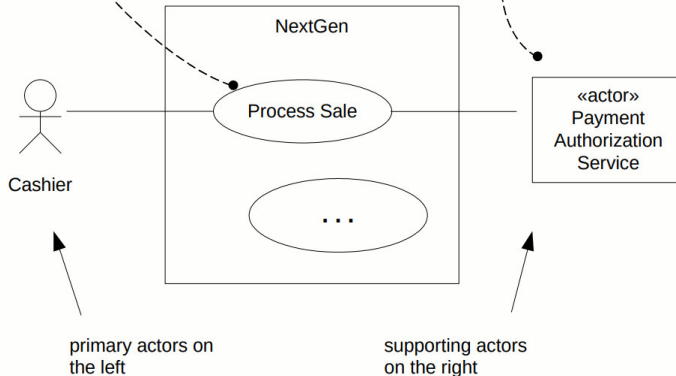


(adopted from "Applying UML and Patterns" by Craig Larman)

UML Use Case Diagrams – Recommendations

For a use case context diagram, limit the use cases to user-goal level use cases.

Show computer system actors with an alternate notation to human actors.



(adopted from "Applying UML and Patterns" by Craig Larman)

Requirements Effort through UP Disciplines

Discipline	Artifact	Comments and Level of Requirements Effort				
		Incep 1 week	Elab 1 4 weeks	Elab 2 4 weeks	Elab 3 3 weeks	Elab 4 3 weeks
Requirements	Use-Case Model	2-day requirements workshop. Most use cases identified by name, and summarized in a short paragraph. Pick 10% from the high-level list to analyze and write in detail. This 10% will be the most architecturally important, risky, and high-business value.	Near the end of this iteration, host a 2-day requirements workshop. Obtain insight and feedback from the implementation work, then complete 30% of the use cases in detail.	Near the end of this iteration, host a 2-day requirements workshop. Obtain insight and feedback from the implementation work, then complete 50% of the use cases in detail.	Repeat, complete 70% of all use cases in detail.	Repeat with the goal of 80–90% of the use cases clarified and written in detail. Only a small portion of these have been built in elaboration; the remainder are done in construction.
Design	Design Model	none	Design for a small set of high-risk architecturally significant requirements.	repeat	repeat	Repeat. The high risk and architecturally significant aspects should now be stabilized.
Implementation	Implementation Model (code, etc.)	none	Implement these.	Repeat. 5% of the final system is built.	Repeat. 10% of the final system is built.	Repeat. 15% of the final system is built.
Project Management	SW Development Plan	Very vague estimate of total effort.	Estimate starts to take shape.	a little better...	a little better...	Overall project duration, major milestones, effort, and cost estimates can now be rationally committed to.

(adopted from “UML 2 and the Unified Process” by Arlow&Neustadt)

An Example of Use Cases Classification

Fully Dressed	Casual	Brief
Process Sale Handle Returns	Process Rental Analyze Sales Activity Manage Security ...	Cash In Cash Out Manage Users Start Up Shut Down Manage System Tables ...

(adopted from "UML 2 and the Unified Process" by Arlow&Neustadt)

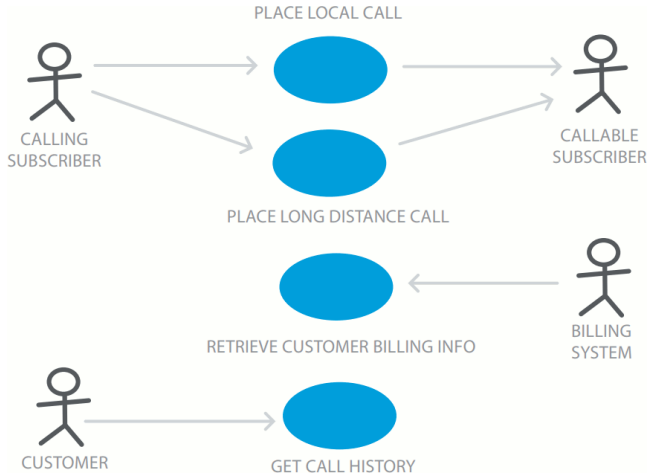
Use Case 2.0 by Jacobson et al.

(see “Use Case 2.0: The Guide to Succeeding with Use Cases” by Jacobson&Spence&Bittner)

- To link the use cases and user stories.
(“the stories cover how to successfully achieve the goal, and how to handle any problems that may occur on the way”)
- It introduces “use-case slice” to map use cases to development.
(“the use cases are sliced up to provide suitably sized work items, and where the system itself is evolved slice by slice”)
- A use-case slice is a set of one/more user stories related to the use case.
(that will be developed together in one increment with defined value to an end-user)
- With Use Case 2.0, the use case analysis is
 - lightweight,
(in both its definition and application)
 - scalable,
(and suitable for teams and systems of all sizes)
 - versatile,
(and suitable for all types of systems and development approaches)
 - and easy to use.
(UC models can be quickly set up and the slices created by the teams needs)



An Example of Use Case 2.0



(adopted from "Use Case 2.0: The Guide to Succeeding with Use Cases" by Jacobson&Spence&Bittner)

An Example of Basic and Alternative Flows

BASIC FLOW

1. Insert Card
2. Validate Card
3. Select Cash Withdrawal
4. Select Account
5. Confirm Availability
of Funds
6. Return Card
7. Dispense Cash

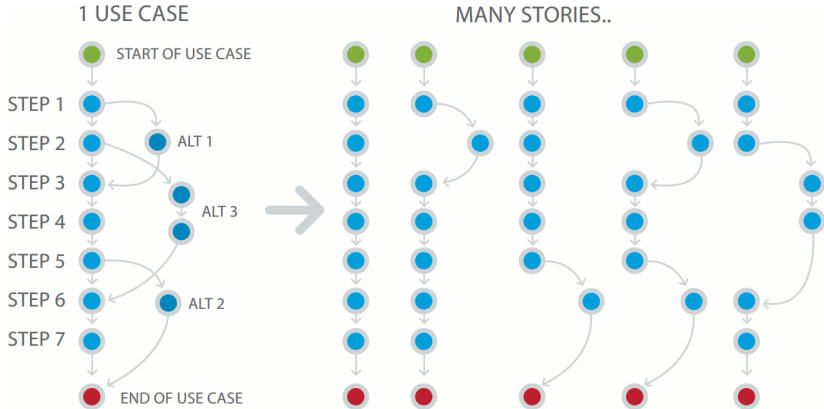
ALTERNATIVE FLOWS

- A1 Invalid Card
- A2 Non-Standard Amount
- A3 Receipt Required
- A4 Insufficient Funds in ATM
- A5 Insufficient Funds in Acct
- A6 Would Cause Overdraft
- A7 Card Stuck
- A8 Cash Left Behind
- etc..

(adopted from "Use Case 2.0: The Guide to Succeeding with Use Cases" by Jacobson&Spence&Bittner)

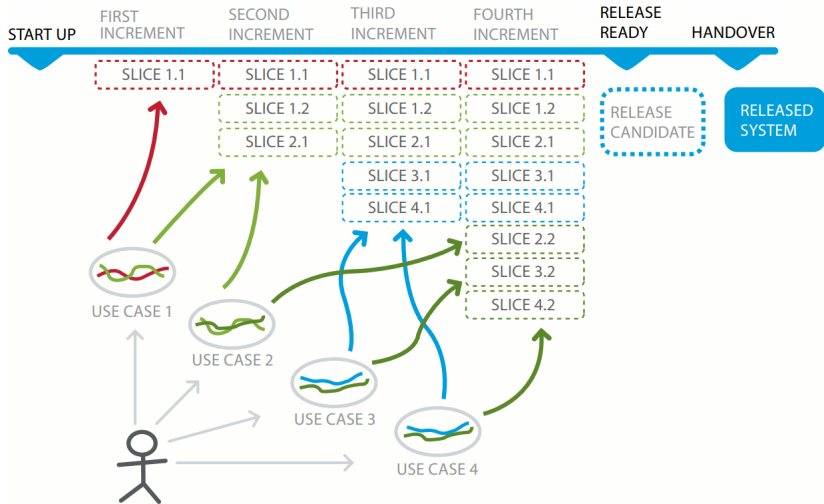


An Example of Use Cases and User Stories



(adopted from "Use Case 2.0: The Guide to Succeeding with Use Cases" by Jacobson&Spence&Bittner)

Use Cases, Slices, Increments, and Releases



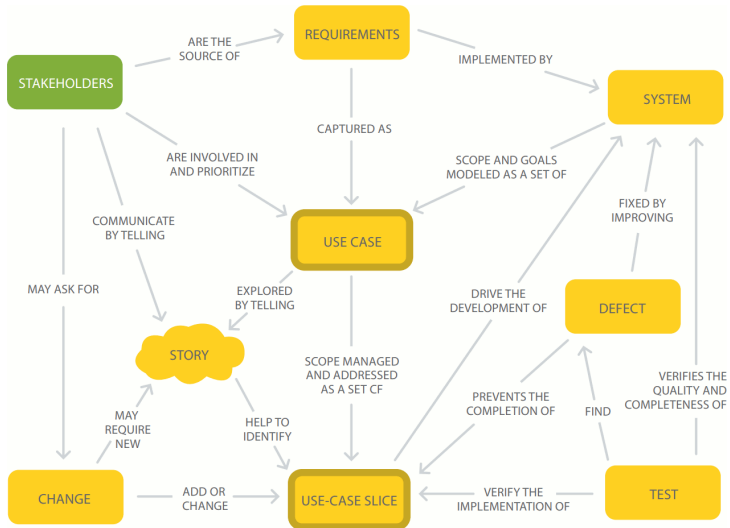
(adopted from "Use Case 2.0: The Guide to Succeeding with Use Cases" by Jacobson&Spence&Bittner)

A Example of a Use Case and its Slices

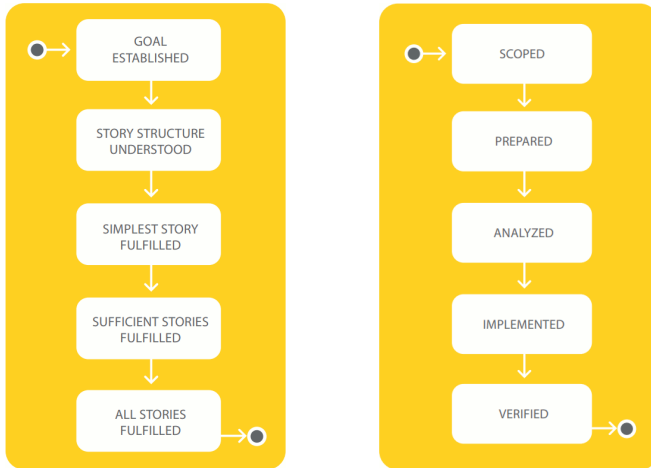


(adopted from “Use Case 2.0: The Guide to Succeeding with Use Cases” by Jacobson&Spence&Bittner)

Use-Case 2.0 Concept Map



Life-cycle of a Use Case and a Slice



(adopted from "Use Case 2.0: The Guide to Succeeding with Use Cases" by Jacobson&Spence&Bittner)

The diagram illustrates the relationships between various software development artifacts. The artifacts are represented by boxes of different shapes and colors: yellow for high-level artifacts (Requirements, System, Use Case, Use-Case Slice, Test), blue for detailed artifacts (Use-Case Model, Use-Case Narrative, Use-Case Realization, Test Case), and a grey box for a summary statement.

Artifacts and their components:

- REQUIREMENTS** (Yellow box): Visualized as **USE-CASE MODEL** (blue box with icon). Captured as **USE CASE** (yellow box).
- SYSTEM** (Yellow box): Implemented by **REQUIREMENTS**. Explains how requirements are handled by **USE-CASE REALIZATION** (blue box with icon).
- USE CASE** (Yellow box): Defines and contextualizes **USE-CASE NARRATIVE** (blue box with icon). Described by **STORY** (yellow cloud). Scope managed and addressed as a set of **USE-CASE SLICE** (yellow box).
- USE-CASE NARRATIVE** (Blue box): Detailed by a set of **USE-CASE MODEL**. Outlines and influences **STORY**. References **SUPPORTING INFORMATION** (blue box with icon).
- USE-CASE SLICE** (Yellow box): Assigned to **STORY**. Completed by **TEST CASE** (blue box with icon). Verified by executing **TEST CASE**.
- STORY** (Yellow cloud): Explored by telling **USE CASE**. Assigned to **USE-CASE SLICE**. Completed by **TEST CASE**. Influences **USE-CASE NARRATIVE** and **SUPPORTING INFORMATION**.
- TEST CASE** (Blue box): Executed as part of a **TEST** (yellow box). Verifies the quality and completeness of **SYSTEM**.
- TEST** (Yellow box): Executed as part of a **TEST CASE**.
- USE-CASE REALIZATION** (Blue box): Explains how requirements are handled by **SYSTEM**. Impact on the system explained by **USE-CASE SLICE**.
- SUPPORTING INFORMATION** (Blue box): References **USE-CASE NARRATIVE**. Complemented by **USE-CASE MODEL**.

Summary Statement (Grey box):

FLOWS, SPECIAL REQUIREMENT AND SYSTEM-WIDE REQUIREMENTS ARE GROUPED INTO STORIES AND ASSIGNED TO SLICES FOR ELABORATION, ANALYSIS, DESIGN AND TESTING.. THE STORIES CAN BE EXPLICITLY LISTED AS PART OF THE USE-CASE NARRATIVE OR TACTICALLY IMPLIED BY THE USE-CASE SLICES.



Supplementary Specification or Requirements

- Another non-functional and out of scope requirements, e.g.,
 - those in URPS+ classes of requirements,
 - or functionality going through multiple use cases.
- For example,
 - usability (e.g., “Text will be visible from the distance of 1 meter.”),
 - technical (e.g., open-source, or cannot use open-source parts),
 - application specific requirements, etc.

An Example of Supplementary Requirements

ID	Rule	Changeability	Source
RULE1	Purchaser discount rules. Examples: Employee—20% off. Preferred Customer—10% off. Senior—15% off.	High. Each retailer uses different rules.	Retailer policy.
RULE2	Sale (transaction-level) discount rules. Applies to pre-tax total. Examples: 10% off if total greater than \$100 USD. 5% off each Monday. 10% off all sales from 10am to 3pm today. Tofu 50% off from 9am-10am today.	High. Each retailer uses different rules, and they may change daily or hourly.	Retailer policy.
RULE3	Product (line item level) discount rules. Examples: 10% off tractors this week. Buy 2 veggieburgers, get 1 free.	High. Each retailer uses different rules, and they may change daily or hourly.	Retailer policy.

(adopted from “UML 2 and the Unified Process” by Arlow&Neustadt)

Requirements Artefacts in Iterative Methods

Discipline	Artifact Iteration→	Incep.	Elab.	Const.	Trans.
		I1	E1..En	C1..Cn	T1..T2
Business Modeling	Domain Model		s		
Requirements	Use-Case Model	s	r		
	<i>Vision</i>	s	r		
	<i>Supplementary Specification</i>	s	r		
	<i>Glossary</i>	s	r		
	<i>Business Rules</i>	s	r		
Design	Design Model		s	r	
	SW Architecture Document		s		
	Data Model		s	r	

Table 7.1 Sample UP artifacts and timing. s - start; r - refine

(adopted from "Applying UML and Patterns" by Craig Larman)

Requirement Elicitation Techniques

Traditional:

- interviews,
- surveys and questionnaires,
- task&domain analysis,
- observation, etc.

Modern:

- prototyping,
- brainstorming,
- Joint Application Development (JAD),
- workshops.

Interviews

- With end-users and a domain expert.
- Can be structured (formal) or unstructured (informal).
 - the structured is with predefined questions,
(there can be also predefined answers to select, or not)
- It is necessary to use unbiased questions.
- Participants must be patient.

Surveys and Questionnaires

- Usually combined with the interviews.
- It is a passive technique (without a direct interaction).
 - + there is time to answer, even repeatedly
 - it is difficult to explain/particularise questions and answers
- Questions with prepared answers or score scales.
(single or multiple-choice questions, answers with comments)

Observation

- If a user cannot provide enough information.
- Three types of the observation:
 - passive – by an observer or a video-camera
 - active – an analyst participate on the observed task
 - demonstrative – a user explains the task
- Under the observation, users tend to behave differently.
- It should be long term, in different times, various users, etc.
(however, it is expensive to process such a lot of observation data)

Analysis of Task/Domain Documents and Systems

- The analysis can be focused on particular tasks or on a domain.
- There are always some documents to analyse.
(a new software system will be implementing already existing tasks)
- Task analysis from internal documents, reports, forms, requests, etc.
(describing, e.g., internal structure, processes, their artefacts, resources, etc.)
- A domain is described by practices, policies, books, other systems, etc.
- The analysis is much easier in process-driven organisations.
(in the sense of Capability Maturity Model/CMM, where processes are described)

Prototyping

- To get fast and reliable feedback from end-users.
- A best practice in iterative software development processes.
- Different types of software prototypes – usually throw-away.
(types: throw-away, evolutionary, low/high fidelity; see a previous lecture)
- It may be quite expensive, usually just for
 - requirements that cannot be obtained by other means,
 - in the case of conflicting requirements,
 - if there is a communication issue between an analyst and end-users.
- Rapid Application Development (RAD) tools can help.
(to easy build both low-cost drawing-based wire-frames, or more time-consuming code-based prototypes that can be partially reused in further development)

Brainstorming

- Utilised to get new ideas and for quick problem-solving.
(especially to get “out-of-box” solutions that would be difficult otherwise)
- To find a compromise solutions from many users with various, often conflicting, requirements.
- A moderated discussion, e.g., the nominal group technique:
 - 1 A facilitator presents a question/problem.
 - 2 Participants are asked to write their ideas/solutions anonymously.
 - 3 The facilitator collects the ideas and the group votes on each idea.
(to “distil” the best ideas/solutions).
 - 4 The top ranked ideas/solutions may be sent back to the group or to subgroups for further brainstorming.

Joint Application Design/Development (JAD)

- A methodology developed by Arnie Lind in IBM Canada for
 - requirements gathering and negotiations
 - and user interface design.
- It is based on the following ideas:
 - Users who do the job have the best understanding of that job.
 - Developers have the best understanding of how technology works.
 - Business and SW development processes work the same basic way.
 - The best software comes out of a process that all groups work as equals and as one team with a single goal that all agree on.
- Participants (key people) design together a product concept.
(facilitator, sponsor, 3 to 5 end-users, executives, developers/observers, scribes)
 - the sponsor makes final decisions regarding go or no-go direction
- No more than 8 people (15 people for skilled teams).
- To specify a systems details from the business perspective.
(not from a technical perspective)



Thank you for your attention!

Marek Rychlý

`<rychly@fit.vutbr.cz>`