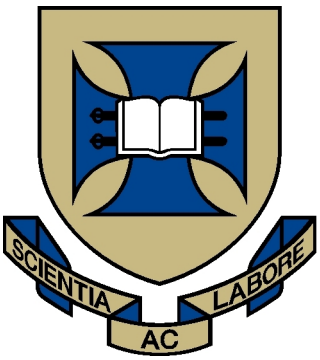
Daniel **Fitz**
43961229



University of Queensland

CSSE3002 – The Software Process

Lecture Notes

Table of Contents

Software Engineering

- Application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.
  - That is, the application of engineering to software.
  - IEEE Standard 610.12-1990
- Concerned with theories, methods and tools that enable professional software development.

“The topic that we call software engineering is both exciting and frustrating. Exciting because it draws on many technical disciplines and provides a harness that binds each discipline to the next. Frustrating, because it demands knowledge in a multitude of topic areas and seems to be infinitely expandable.” - Roger Pressman, 1992

Software Engineering Process

A structured set of activities followed to develop a software system

- Tools
- Methods
- Practices

Well Engineered Software

- Usable
- Dependable
- Maintainable
- Efficient
- How do costs come into this?
  - Trade-offs may be involved
    - appropriate
    - cost-effective

Process Models

- Abstract representation of a process
- Plan Driven
  - Structured / Traditional
- Incremental
- Agile
- Lean
- Formal

Plan Driven Processes

Waterfall

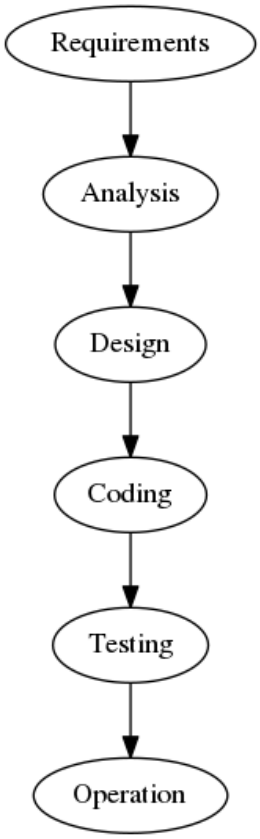


Figure 1: Diagram explaining Waterfall

- Introduced iteration between phases
- Prototyping
  - Requirements
  - Design

VModel

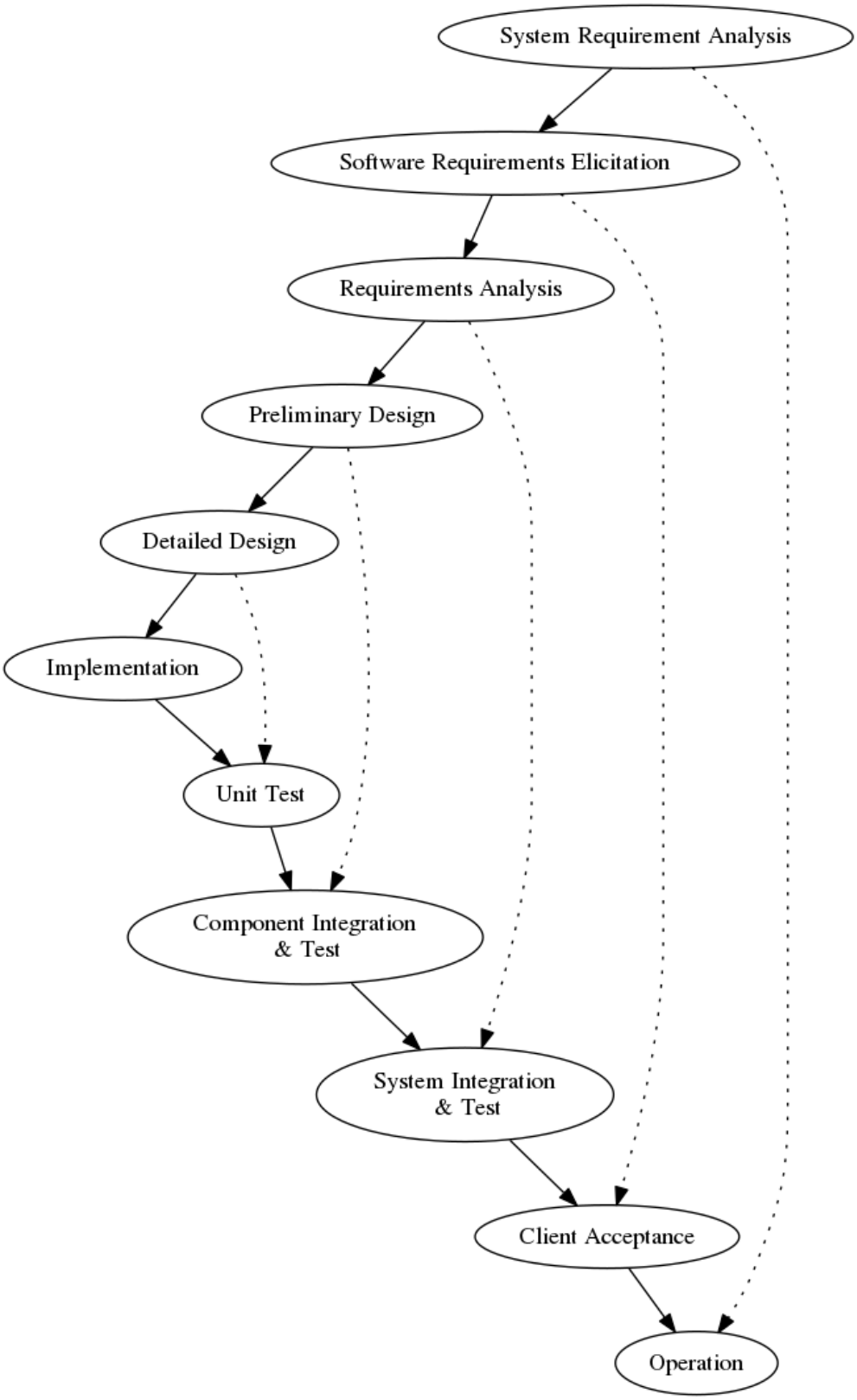


Figure 2: Diagram explaining V Model

Spiral

- Focus on process control
- See

`http://csse.usc.edu/TECHRPTS/1988/usccse88-500/uscsse88-500.pdf`

Incremental Processes

Unified Process

Unified Process is allied closely with UML

- Four distinct phases
  - Inception, Elaboration, Construction and Transition
- Considers activity balance across workflows and phases

OPEN

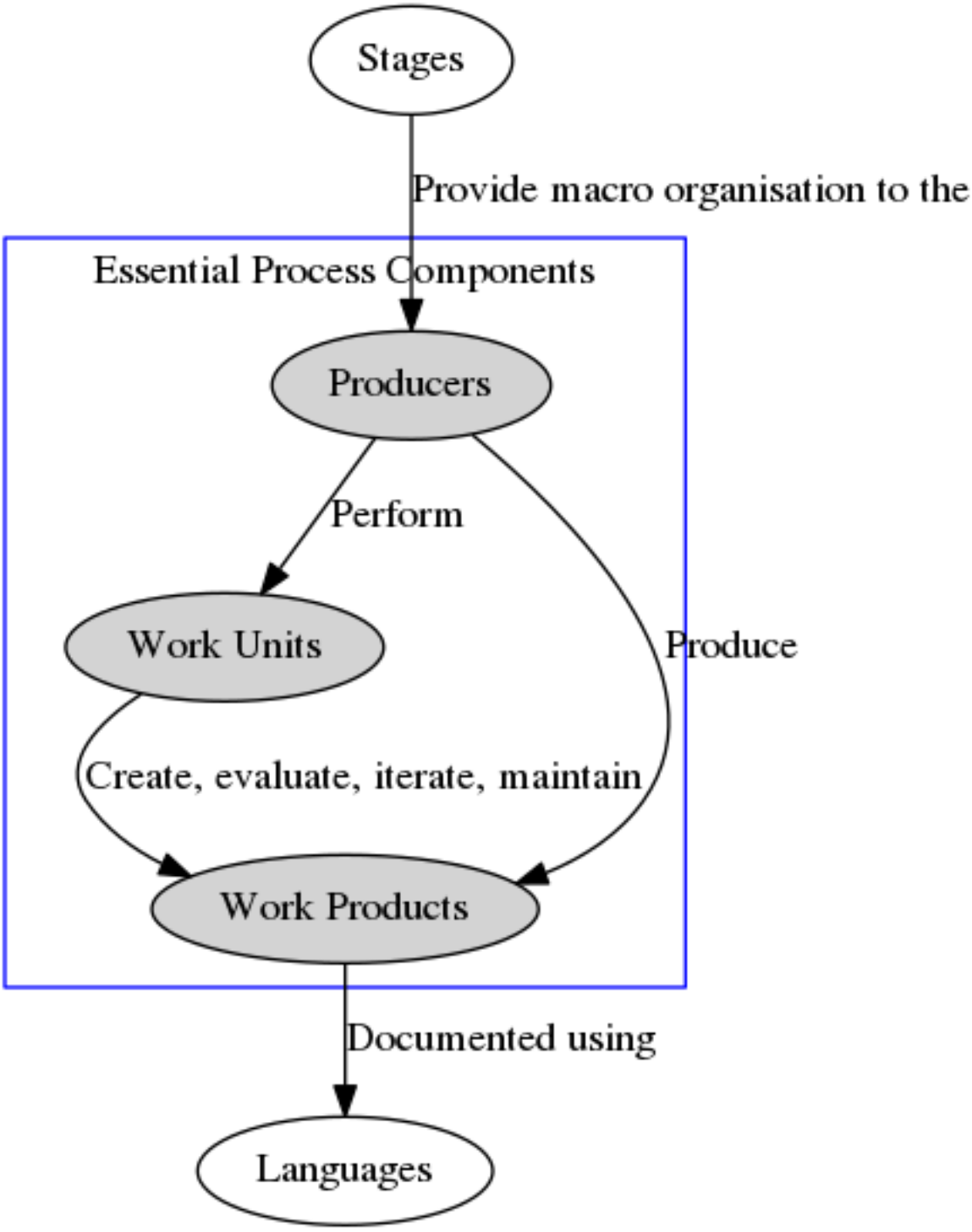


Figure 3: Diagram of OPEN Process

- Process framework
  - process is instantiated from the framework
  - metamodel documents the framework
- Contracts between components
  - process construction

  - scheduling

Agile Processes

- Scrum, XP, FDD, DSDM
- Embrace change
  - Requirements are never fixed
  - Stop pretending and get used to it
- Deliver early and deliver often
  - A working system delivers value
  - A deployed system generates revenue

Lean Development

- More a philosophy than a process
  - Think Big
  - Act Small
  - Fail Fast
- Eliminate Waste
- Amplify Learning
- Decide as Late as Possible
- Deliver as Fast as Possible
- Empower the Team
- Build Integrity In
- See the Whole

Formal Processes

- Application of mathematical formality to software development
  - formal specification
  - transformation of specification to code

Process

- All SE Processes involved phases
  - Requirements
  - Design
  - Development (implementation, coding)
  - Testing (Verification)
  - Delivery and Maintenance
- These are never disjoint, never just sequential
- We iterate between them, and we blur the distinctions because we want to get it right
- Software Engineering cannot work without a defined development process
  - anything else is randomised hacking
- Processes cannot work if they are not usable
  - people don't read telephone books cover to cover
- Good processes should engage the team
  - support technical excellence and innovation
  - embed a culture of trust and responsibility

Standards

- Rules, guidelines and heuristics
- De facto – implicit agreement
  - easily changed

- De jure – formal agreement
  - usually debated and documented

Standard Adoption

- Voluntary
  - achieving good practice
  - safety net
- Required
  - demands of clients
  - certification requirements
  - follow on from other standards
  - process improvement activity

SE Standards

- Normative and informative
- Document centred
- Adaptable

Main SE Standards

- ISO/IEC 12207:2008
  - Systems and software engineering – Software life cycle processes
- ISO/IEC/IEEE 15288:2015
  - Systems and software engineering – System life cycle processes
- ISO/IEC/IEEE 15289:2015
  - System and software engineering – Content of life-cycle information items (documentation)

ISO/IEC 12207

- Framework for lifecycle modelling
- Focus on bespoke software
  - including product and services
- Includes process for defining, controlling and improving software processes
- Last reviewed in 2013

ISO/IEC/IEEE 15288

- Framework for process descriptions
- Focus on system engineering
  - software as a component of system
- Focus on bespoke system development
- Includes process for defining, controlling and improving processes
- Ratified in 2015

12207 vs 15288

- 15288 focuesses on systems
  - hardware, software, people, facilities, material, …
- 12207 focusses on software
  - intended to be used for software component of 15288

ISO/IEC/IEEE 15289

- Standard project documentation
- Focus on purpose and content
  - not necessarily a formal document (e.g. central data repository)
- Ratified in 2015

ISO/IEC 29110

- Software engineering – Lifecycle profiles for Very Small Entities (up to 25 people)
- Subset of 12207 and 15289
- Profiles for different scales of complexity
  - component of a system

  - up to multiple commercial projects
- Ratified in 2016

IEEE Standards

- Terminology
- QA Plans
- Configuration Management
- Requirements Specification
- Unit Testing
- V&V
- Reviews & Audits
- Productivity Metrics
- Quality Metrics
- Project Management Plans
- User Documentation
- Maintenance

Ethics

Code of Ethics

- Agreed standard of behaviour
- Mark of professionalism
- most professional bodies have one
- Enforceable?

Australian Computer Society (ACS)

Primary of Public Interest place interests of public above personal, business or sectional interests
Enhancement of Quality of Life strive to enhance quality of life of those affected
Honesty honest representation of skills, knowledge, services and products
Competence work competently and diligently for stakeholders
Professional Development enhance your own development and your colleagues and staff
Professionalism enhance integrity of the ACS and respect of members for each other

What is Requirements Engineering

- Requirements engineering is a term often used for a systematic approach to acquire, analyse, validate, document and manage requirements
- Typically implemented as a cyclic or iterative process
- Requirements validation may include prototype construction and evaluation
- Applied at both system and software levels, often with interleaved system architecture design

What is a Requirement?

1. A condition or capability needed by a user to solve a problem or achieve an objective
2. A condition by a system or system component to satisfy a contract, standard, specification or other formally imposed document
3. A documented representation of a condition or capability as in 1 or 2

- There is a relationship between the quality / cost / timeliness / etc. of the product and the quality of the process
- Both functional and non-functional requirements are essential for successful software
  - Both must be verified
    - Consequently they must be testable
      - Non-functional Requirements should be measurable

Requirements Engineering Products

- Primary outcome is a requirements specification
  - Essentially a contract between user and developer
  - Basis for all subsequent development and verification processes
- Secondary outcome is usually system and software acceptance test criteria

Why is RE important?

- Most faults observed in a software project are from incorrect, incomplete, or misinterpreted functional specifications or requirements
- Helps earlier detection of mistakes, which are much more costly to correct if discovered later
- Forces clients to articulate and review requirements
- Enhances communications between participants
- Helps record and refine requirements
- All this is about producing good requirements

Advice/Perspective

- … a systematic approach to finding, documenting, organizing, and tracking the changing requirements of a system
- In practice it is impossible to produce a complete and consistent requirements document
- Getting the requirements right is critical for success
- Requirements are rarely right at the start of a large project
  - Expect change
  - Manage it
  - Agile mantra “Embrace Change”

Functional Requirements

- Requirements (or capabilities) for functions (specific behaviour) that must be performed by the system
  - e.g. read a bar code, change a user name
- Primary focus of most requirements activities

Non-Functional Requirements

- Constraints on performance or quality

Product Properties

- Requirements on the behaviour of the product
  - System shall process a minimum of 8 transactions per second
  - User credit card details shall be secured

Process Properties

- Requirements on the practices used to develop / produce the system
  - Control software shall be verified in accordance with IEEE STD 1012-1998

Classification of Non-Functional Requirements

According to the ISO standard

- Safety requirements
- Security requirements
- Interface requirements
- Human engineering requirements
- Qualification requirements
- Operational requirements
- Maintenance requirements
- Design constraints

Sources of Requirements

- Users
  - e.g. customers or end-users – user requirements
- Other Stakeholders
  - e.g. marketing experts, regulators, managers, business owners, developers
- Non-Human Sources

- e.g. other devices or systems in the environment

The Requirements Engineering Process

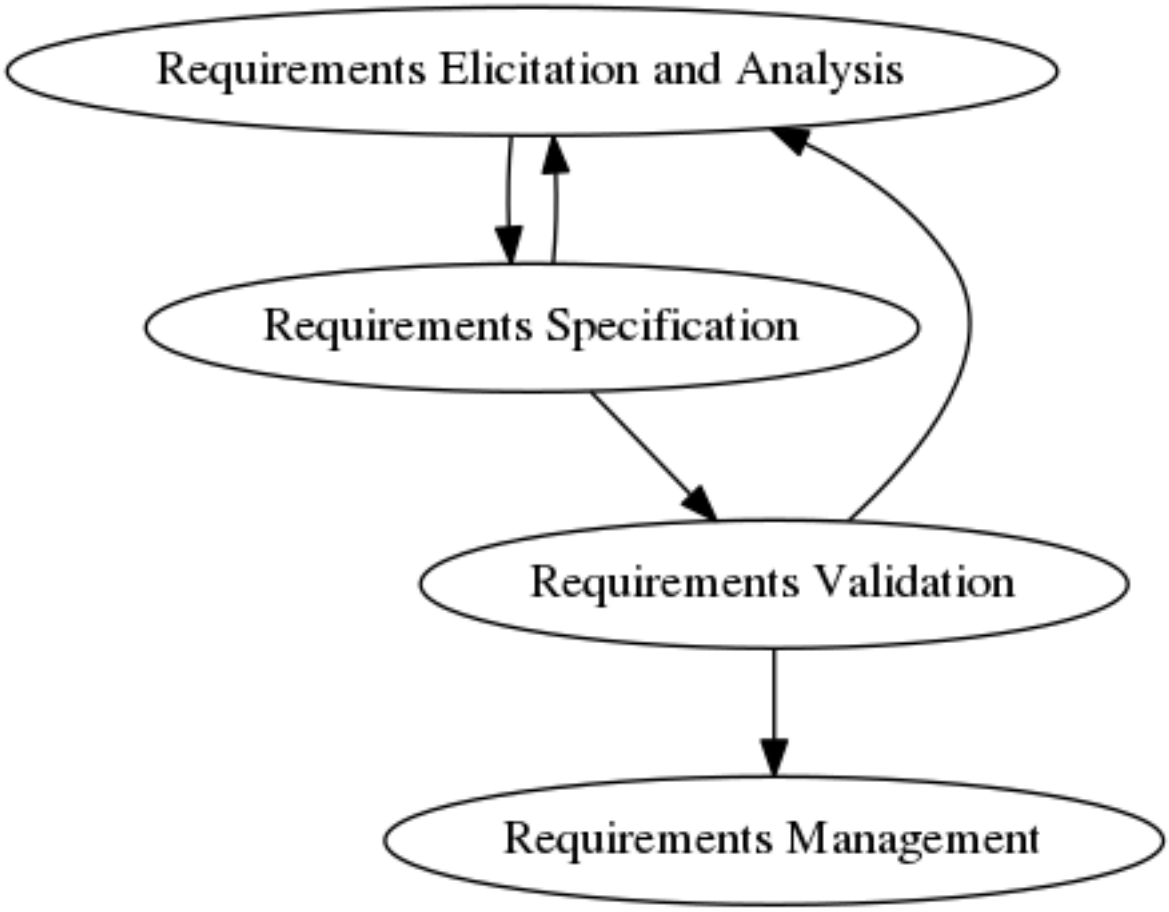


Figure 4: Requirements Engineering Process

Summary

- Requirements engineering is concerned with eliciting, analysing, documenting, validating and managing requirements
- The process, and the specification it produces, is the primary link between the user and the system developer
- The requirements specification is also the basis for all subsequent development activity, and must guide all decisions that determine the resultant product quality
- Requirements engineering is the key to product quality

Project Charter

Vision Statement

- Long term purpose of system
  - a bit idealistic
- For – target audience
- Who – statement of need
- The – product
- Is – category
- That – reason to use
- Unlike – alternative

- Out Product – advantage

Goals

- High-level
- What project will accomplish

Objectives

- Specific
- Supports a goal
  - think “how” it does this
- Describe with an action verb
  - measurable
  - address project end result

SMART

- Specific
  - what is to be accomplished
  - only essential aspects
- Measurable
  - need success/completion criteria
- Agreed-upon
  - common understanding amongst stakeholders
- Realistic
  - achievable with available resources
- Time-based
  - realistic deadline

Business Benefits

- High-level but concrete
  - Increased revenue
  - Reduced costs
  - Improved efficiency
  - Improved customer satisfaction
  - …

Scope

- What is to be delivered
  - by end of project
  - releases determined later
- What is explicitly out of scope
  - does not relate to business benefit

Stakeholders

- Sponsor
- Influencers
- Users
  - key
  - restricted
  - super
- Anti-Users
- Others
  - e.g. infrastructure team

Assumptions

- Expected to occur

Constraints

- Restrictions

- project
- development team

Business Value Not System Requirements

Understanding the Business

- Developers and Stakeholders need a shared understanding of the project's purpose
  - easier when they collaborate continuously
- Focus on value to be delivered
  - not just the requirements
- Enables better decisions, designs and suggestions
  - developers are part of the value chain
    - not just serving it

Business Model

A business model describes the rationale of how an organisation creates, delivers, and captures value

Business Model Canvas

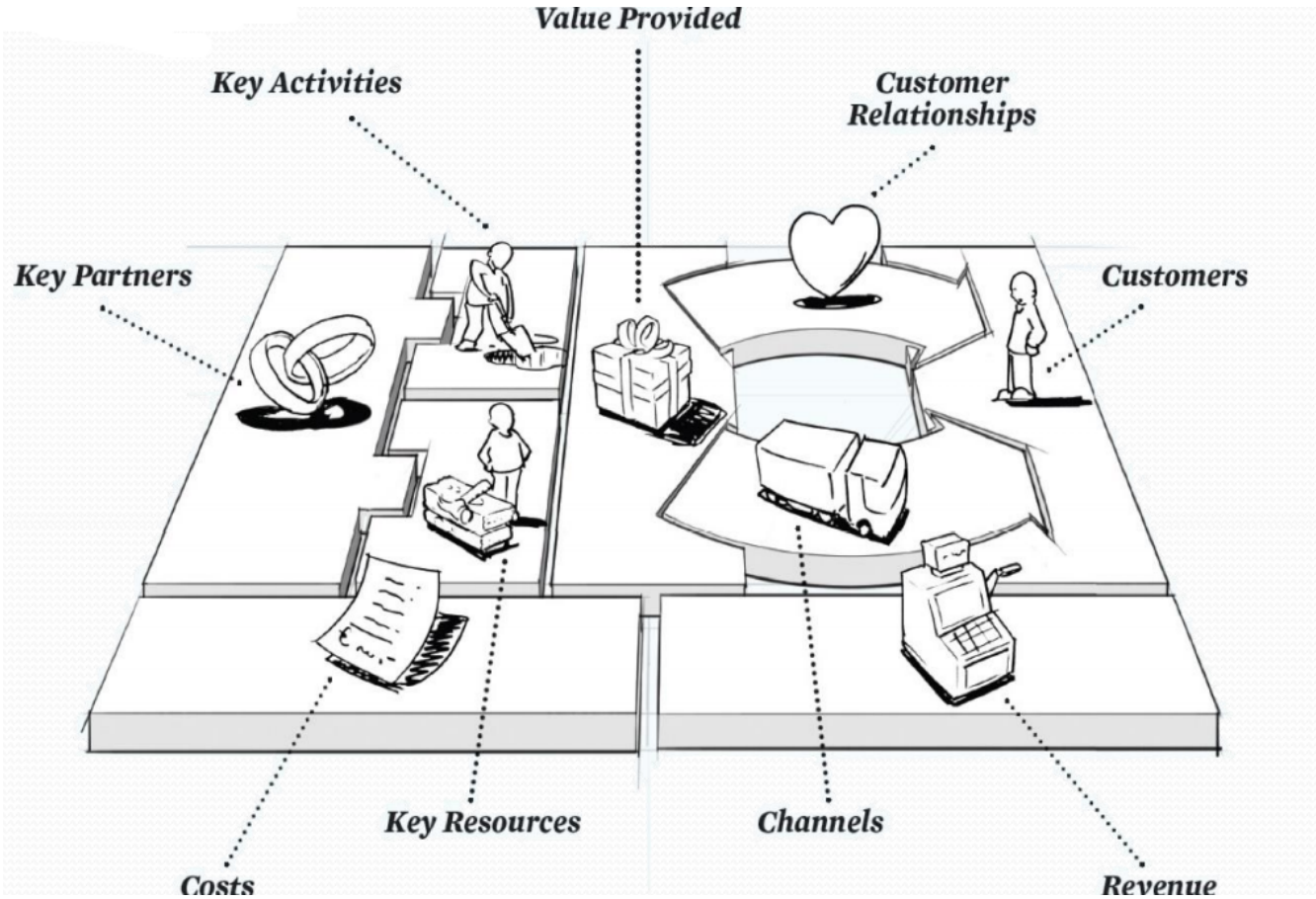


Figure 5: Business Model Canvas

Customers

- Personas
- Who's Impacted

- Stakeholders

Value Proposition
- Use Cases
- Specification by Example
- Customer Savings / Revenue
- Improvements
- Experience Improvements

Channels
- Systems
- Methods
- Related Features

Relationship
- Direct / Indirect
- Human / Automated
- Assisted / Self Service
- Individually / Collaboratively

Revenue Streams
- Opportunity
- Savings
- Profit
- Improvements

Key Resources
- Systems
  - Primary
  - Secondary / Connected
- Team
  - Development
  - Business

Key Activities
- Use Cases
- Who's Activities
- Connected Activities

Key Partnerships
- Development Team
- Business
- Secondary / Connected Teams
- Impacted Teams
- Related Teams

Cost Structure
- Opportunity
- Development Estimates
- Quantity of Customers

Start-Up
- Enthusiastic developers
- No clear business model
  - Or how to monetise success
- Now have focus

- What to out source
- What to develop
- Costs
- Revenue streams

Requirements Elicitation

Elicitation Process

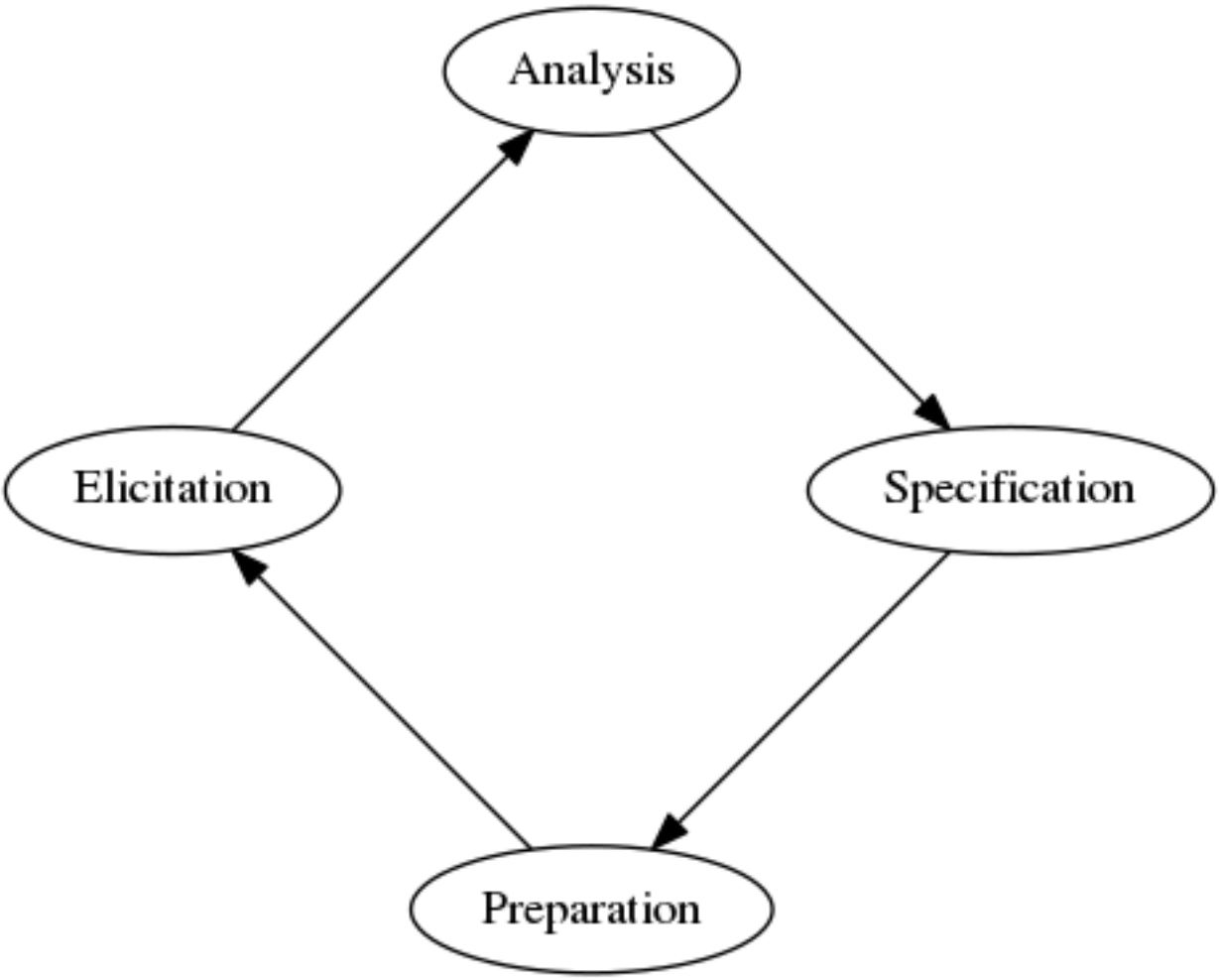


Figure 6: Elicitation Process

Preparation – Sources of Requirements

- Stakeholders
- Users
- Environment
  - application domain
  - organisation
  - operations
    - other system dependencies
      - interface requirements
      - timing constraints

        * execution environment
            - platform
            - reliability and performance
        * criticality
            - mission
            - safety

Know Your Users – User Role Modelling

- What types of people will use the system?
- Don't think of an anonymous user
- Identify different user roles
    - brainstorm initial set
    - group related roles
    - consolidate roles
    - refine roles
- Don't get stuck on organisational roles

Elicitation Challenges

- Stakeholders and users may not be able to describe their tasks well
    - make assumptions and leave things unstated
- No-one knows everything
- Requirements conflict
- Implicit requirements

Elicitation Techniques

Interviews

- Effective for understanding problem and eliciting *general* requirements
- Prepare questions in advance
    - discussion needs a starting point
    - primarily open-ended questions
    - strawman model if you have some data
- Suggest ideas and alternatives
    - users may not realise what is possible
- Active listening
    - paraphrase what you understand
- Clarify what's unclear
- Maintain focus

Workshops

- Structured meeting
    - formal roles
    - clear goals
- Multiple stakeholders
    - resolve conflicting requirements
    - quickly gather broad system usage

Focus Groups

- Less structure
    - still need clear goals
- Exploratory discussion
    - needs
    - preferences
    - expectations
- Broad stakeholder representation
- Gather broad-based ideas

Observations
- Observe how users perform their tasks
- Users often cannot describe everything they do
  - too many fine details or habitual tasks
- Time consuming
  - silent observation
  - interactive

Questionnaires
- Inexpensive and easily administered to remote sites
- Collect data from many users
- May feed into interviews or workshops
- Good questionnaires difficult to write

Independent Elicitation Techniques
- Discover information on your own

System Interface Analysis
- Look at other system's functionality
- Data exchange
  - including formats and validation rules
- Services

User Interface Analysis
- Study existing systems
- What should be replicated and avoided
- Good way to learn existing system and processes

Document Analysis
- Business process descriptions
- Existing system documentation
- Industry standards or legislation
- Gain understanding of domain or system

Requirements Modelling

Product vs User Centred

Product-Centred
- Focus on features to be delivered
  - expect users will use features to complete tasks

User-Centred
- Focus on anticipated usage
  - what do users need to accomplish
- Reveal necessary functionality
- Assists with prioritisation

Use Case Modelling
- Models and documents the functional requirements of a problem domain
- Results in the production of the
  - functional requirements
  - which are the detailed, role based, functional account of the requirements

Why Use Cases?

- Formalises users' expectations of what the system is to do and how the system is to be used
- Easy technique to understand
  - documents actual paths through the system
- User-driven process
  - encourages user involvement
- Basis for scoping and prioritising development work
- Basis for acceptance testing
- Well aligned with Business Process Modelling

What is a Use Case?

- A way to use the system
- Externally required functionality
- What the system does
  - not how it does it
- Specify the behaviour of a use case as a flow of events

Actors, Use Cases and Association

Actors things outside of the system that interact with the system
Use Cases features of the system that an actor uses
Associations indicates a relationship between an actor and a use case

Actors

- Everything that interacts with the system
- Not described in detail (they're outside of system)
- Normally act in several use cases
- Represents a role that a user can play
  - many users are represented by a single actor
  - one user can be different actors at different times

Actors are External to System

- Make sure that actors are people or other systems that would actually use the system
- The system does not use itself

Primary and Secondary Actors

- Primary actors are those actors that the system is designed to serve
  - Main users of the system
- Secondary actors are support roles
  - secondary actors only exist so that primary actors can use the system

<<include>> Relationship

- Factor out common behaviour in use cases
  - sequence of steps appearing in multiple use cases
- Scenario always uses the included steps
  - included steps do not have to be a complete use case

<<extend>> Relationship

- Factors out optional behaviour in use cases
  - used when a scenario produces a different result in certain situations
    - when there are optional or uncommon steps that may occur
- The extended scenario may be completed without including the steps from the extending use case
  - removes the need to have many primary scenarios to capture the optional paths of a use case
- Delivery of extending use cases can occur in later phases of development

Extension Point

- Included in the use case description to indicate the point at which the extending use case begins

  - condition that causes the extension is highlighted
- Scenario for the base use case runs as normal until the extension point
- Under certain conditions the extending use case then begins and runs to completion
- Base use case then resumes

Use Cases Limitations

- Interaction focus
  - usage scenarios
- Not suitable
  - batch processing
  - complex business rules
  - computationally intensive
  - real-time systems
  - embedded systems

Activity Diagram

- Shows the steps involved in performing a task
- Special form of state diagram
- Describes a complex task for objects of a class, operations, or use cases
  - focuses on internal processing
  - use where all the events represent the completion of internally generated actions
  - use state diagrams where asynchronous events occur

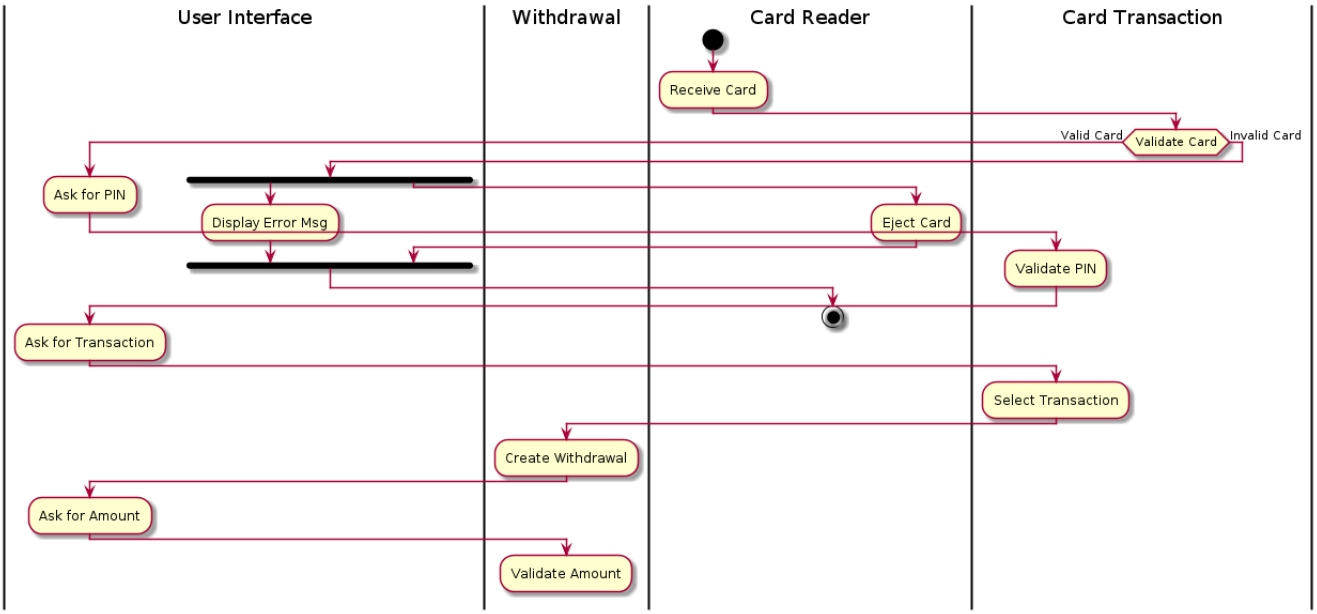


Figure 7: Activity Diagram Example

Activity Diagrams in Use Case Modelling

- Determine triggering event that starts use case flow
- Identify actions and determine control flow
- Add guard conditions and decision points
- Add forking and joining to show parallel activity
- Create invoke activities if complexity requires it

- Group activities into partitions if needed
- Add flows corresponding to alternative scenarios
- Each path should correspond to an individual scenario

User Stories

- Short description of functionality
- From the user's perspective
- Provides value to the user or customer
  - consider both types of clients
- Must be testable
- Provides enough information for developers to make rough estimates