Systems Design and Security

Part 3: Project Management

http://staffwww.dcs.shef.ac.uk/people/A.Simons/

Home ⇒ Teaching ⇒ Lectures ⇒
COM2008/3008

The
University
Of
Sheffield.

Bibliography

- Software Engineering
  - I Sommerville, Software Engineering, 10th ed., Pearson, 2016.
  - R S Pressman and B Maxim, Software Engineering: A Practitioner's Approach, 9th ed., McGraw-Hill, 2019.
- Unified Modelling Language
  - M Fowler, UML Distilled: A Brief Guide to the Standard Object Modelling Language, Addison-Wesley, 3rd ed., 2003.
  - S Bennett, S McRobb R Farmer, Object-Oriented Systems Analysis and Design using UML, 4th ed., McGraw-Hill, 2010.

Outline

- Managing customers, developers
- Managing development risks
- Understanding systems and software systems
- Requirements engineering process
- Psychological and socio-political issues
- Modelling and coordination

Reading: Sommerville chapters 4-5, 22-23;
Pressman chapters 5, 24-27;
Bennett, et al. chapters 2-3, 21

Project Management

- Managing the “four Ps”
  - people – customers, developer teams
  - product – software system, context
  - process – lifecycle, analysis and design
  - project – deadlines, deliverables, risks
- Key issues
  - mitigate risks of failure
  - satisfy customer expectations
  - deliver on time and within budget

People Management

- Customer and stakeholders
  - customer – commissions the system
    - may be a business manager, company IT director
  - stakeholders – have vested interests
    - business managers – business policies, goals
    - business end-users – usability of the system
    - third-party customers – source of business revenues
- Developer teams
  - senior managers – run the software house
  - technical managers – manage individual projects
  - developers – produce designs, code

Business Stakeholders

- Definition of "Customer"
  - your client, the person who commissions the system – you must meet his/her objectives
  - but he/she may not be the final end-user – may not understand all the operational issues
- Definition of "Stakeholders"
  - someone with a vested interest in how the system will work: manager, operator, IT support, beneficiary
  - conflicting interests in how the system works – different preferences, or outright resistance!
  - need to balance interests of all these parties – deal with socio-political aspects, such as balance of power in the workplace
  - need to manage expectations – what is possible, desirable, impossible to deliver

Software Developers

- Required skills
  - technical – design, coding accuracy
  - communication – within team, with customer
  - complementarity – teams need good distribution of skills
    - customer service, analysis and design, coding and testing
- Management issues
  - distribute workload fairly (novice/expert issues)
  - coordinate work increments
    - planning, review meetings; feedback, troubleshooting
    - have long/medium/short-term plans and recovery strategy
    - documentation (models, agendas, minutes, plans, charts)

Product Management

- Scope of the project
  - business context, objectives, what is within/outside scope
  - functional requirements vs performance constraints
  - new build vs extension; standalone vs interoperable system; one-off system vs one in a product line
- Problem decomposition
  - divide and conquer: split into modules and subsystems
  - architecture: distribution over different machines, sites, etc
- Quantitative estimates
  - need exact costing and time information, mostly when solid information is unavailable!
  - eg: COCOMO II [Boehm, 2000] – rigorous cost estimation model
  - eg: function point analysis – lightweight cost estimation model

Function Point Analysis

- Basic idea
  - identify the main business functions – main tasks supported by the system, done by the system's end-users
  - score each function on a scale of 1..3 (easy..hard) based on how difficult to implement, and sum the function points
  - pick a constant and multiply the function points by this constant, to yield the total size (time, cost) of the project
- Difficulties
  - picking accurate size, time, cost constants is a black art!
  - novice developers typically under-estimate time and cost of developing a system by up to a factor of 3!

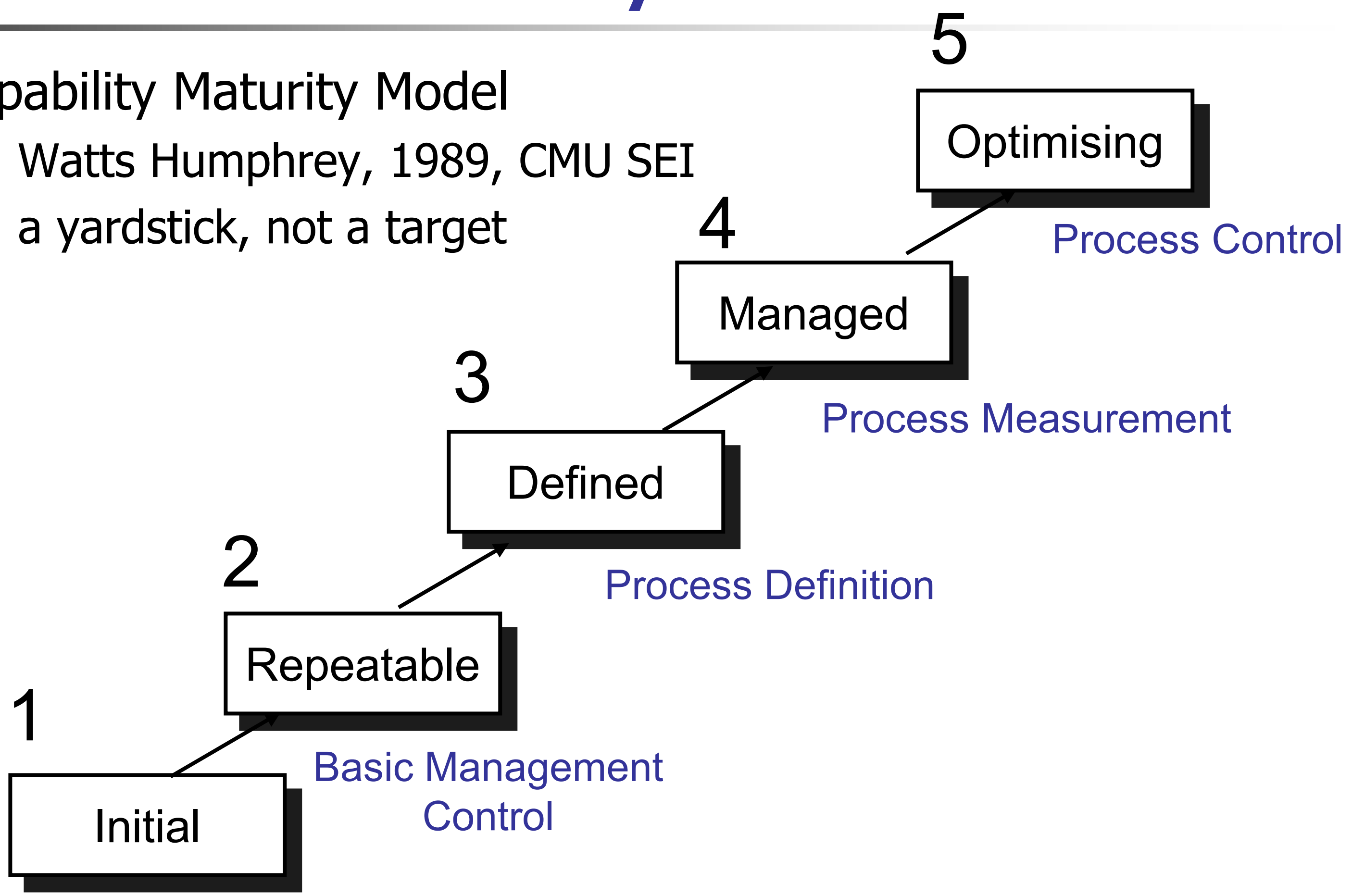
Process Management

- Select a process model
  - to fit the product: wholistic, incremental, unknown
  - to match customer availability: continuous, staged, limited upfront availability
  - to fit the project environment: large scale, coordinated; small scale, prototyping
- Follow the process model
  - plan a set of stages, deliverables
  - modify process to suit project constraints
  - adapt and improve process in the light of experience: Capability Maturity Model [Humphrey, 1989]

Process Maturity

Capability Maturity Model

- Watts Humphrey, 1989, CMU SEI
- a yardstick, not a target



1 Initial

Basic Management Control

2 Repeatable

Process Definition

3 Defined

Process Measurement

4 Managed

Process Control

5 Optimising

Project Management

- Many large software projects
  - are delivered late
  - do not work properly
  - are over budget
  - fail to meet requirements
- Cost of poor quality, USA 2018

  [https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/]
  - Legacy system fixes: $635bn
  - Cancelled projects: $178bn
  - Troubled/delayed projects: $1,275bn
  - Failures: $1.1 trillion, total poor quality: $2.8 trillion!

Risks to Mitigate

- Misunderstand the customer’s needs
- The project scope is poorly defined
- Changes are poorly managed
- The supporting technology changes
- Business goals are changing
- Unrealistic deadlines are set
- Users resistant to new practices (power, control)
- Losing the sponsorship (funding, company champion)
- Unskilled/uncommunicative software team
- Poor management strategy (bid-to-win)

Mismanagement

- Management Failure
  - Bid-to-win/no evidence-based costing strategy
  - No financial or contractual management
  - Failure to understand or manage risk
- Communication Failure
  - Business strategy/processes superseded
  - Costs and benefits of systems not explained
  - Poor customer-developer communication
- Technical Failure
  - No clear requirements definition
  - No control over change requests
  - Wrong architecture/no testing

Charette, R.N.: Why software fails.
IEEE Spectrum, New York, September (2005)

Lab 1: Function Points

- Automated Teller Machine
    - ATM: the “hole in the wall” bank terminal and cash dispenser
    - purpose: to relieve the workload of cashiers (bank “tellers”) and offer customer services

- What are the main function points?
    - Which functions are essential?
    - Which functions are possible?

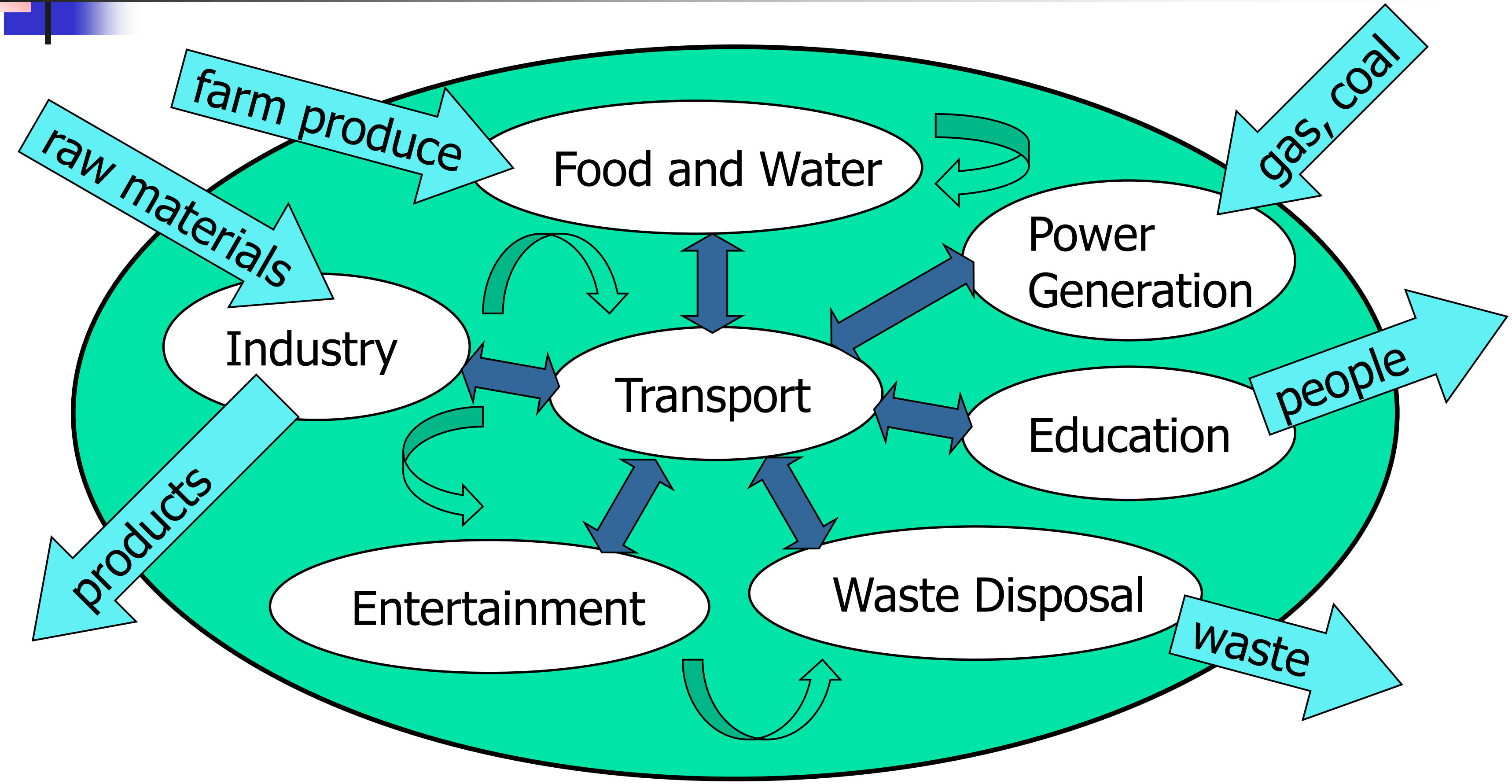
Run a Poll

- Which functions are harder to deliver?
    - Score each function on 1..3 for difficulty
    - Give reasons why some functions are harder/easier

Systems

- A system is a complex whole
  - made up of many collaborating parts, or subsystems
  - the parts all work together to achieve some goal
  - has self-regulating control: feedback/feed forward
  - has emergent (integrative, synergistic) properties
    - "more than the sum of its parts" – not directly dependent
- A system exists in a context
  - separated from the environment by a boundary
  - has inputs from, and outputs to, the environment
- Example systems
  - a city – what are the parts? the goal?
  - the human body – what are the parts? the goal?

City as System



Software Systems

- Exist in a business context
  - goal is to support the business, raise revenues
  - core activity – eg: manufacture, process control
  - subsidiary activity – eg: sales, invoicing, personnel
    - information systems – the majority of software systems
- Interact with other systems
  - interact with physical (paper-based) or human systems
  - interact with legacy (outdated) software systems
- Questions in systems development
  - will the new system bring real benefits?
  - how to integrate with physical/legacy systems?
  - should physical/legacy systems be replaced?

System Requirements

- Functional requirements
  - What will the system do?
  - What must the users accomplish?
- Non-functional requirements
  - How must the system operate?
  - What performance goals are there?
  - What physical constraints exist?
- Usability requirements
  - How easy must the system be to use?
  - What styles of interaction are anticipated?

Functional

- Description of the processing done by the system
  - describe all input and output relationships
  - behaviour for all possible inputs (correct and incorrect)
  - consider all aspects of expected user interaction
  - describe the data that must be produced by the system
- Description of the main functions of the system
  - complete and consistent, expressed in a precise way
  - identify and try to resolve logical inconsistencies, especially in large systems (hard to avoid)
  - may be expressed as essential or desirable requirements

Non-Functional

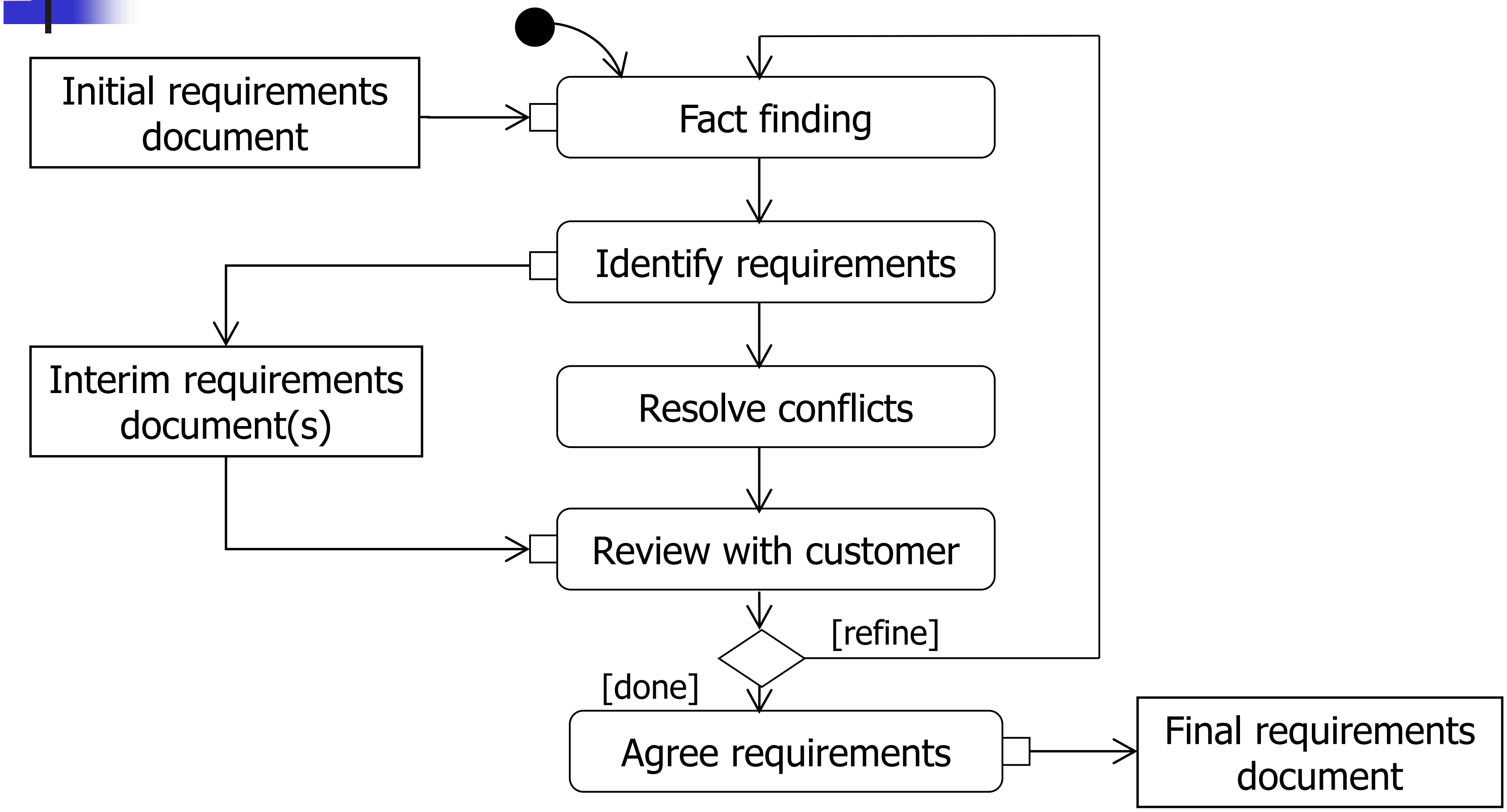
- Product requirements
  - performance (speed, latency), reliability, portability
  - constrained by hardware on which the system will run
- Organisational requirements
  - conform to policies and procedures within the business
- External requirements
  - legal and ethical requirements (security, confidentiality)
  - how the system interacts with other systems

[this classification from Sommerville]

Usability

- User-centred design
  - is the user skill level correctly pitched?
  - what tasks must each user undertake?
  - is the sequence of tasks logical and intuitive?
- Situational factors
  - how ergonomic, fitting with working practices?
  - how robust to incorrect data entry?
- Acceptance criteria
  - how shall we test if a system is a success?
  - is the system a pleasure to use?

Requirements Elicitation



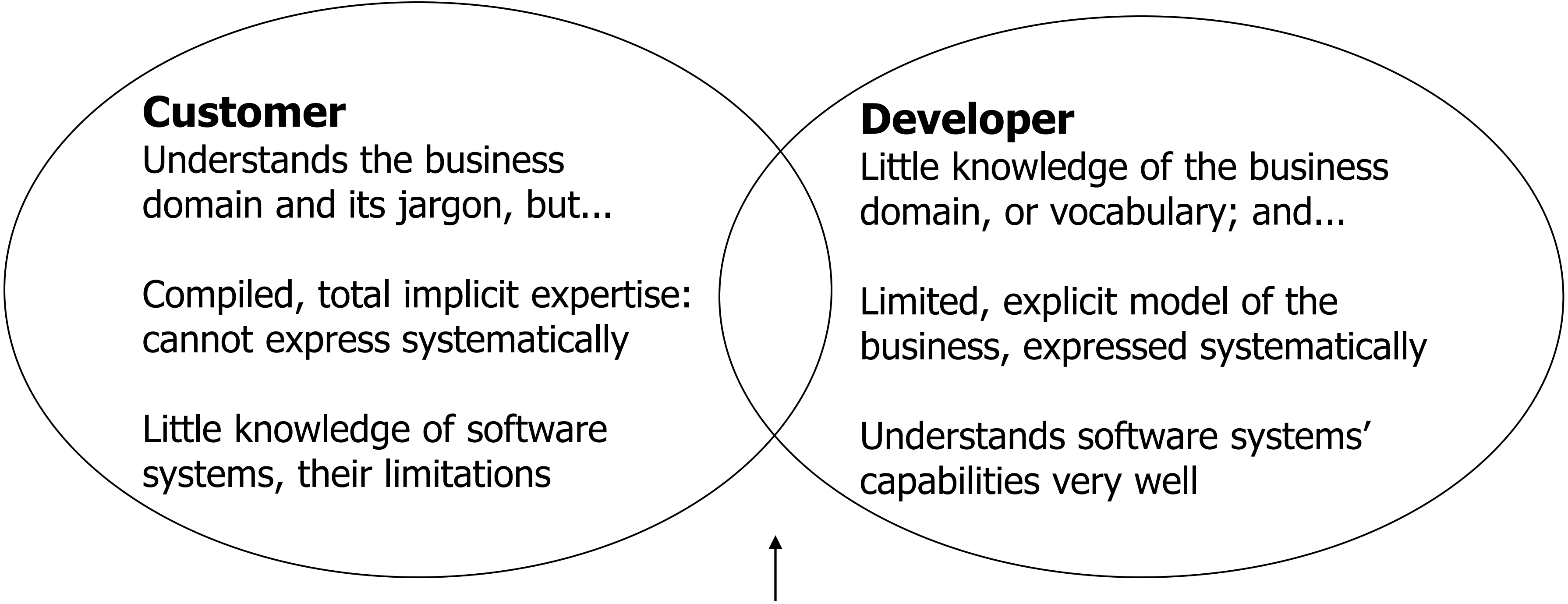
Fact Finding

- Background reading
  - documentation, procedures, job descriptions, and reports
  - provides contextual information – to educate the developer!
- Interviewing/workshops
  - most widely used technique, but requires skill and sensitivity
- Observation
  - get quantitative data, eg: typical time to complete a task
  - useful for observing responses to exceptional situations
- Document sampling
  - paper documents, screenshots: illustrate information flow
- Questionnaires
  - an economical way to collect data, but hard to do well

Starting Points

- Customer may only have a general business vision
  - Opportunity-driven: an idea to capture a new market
  - Necessity-driven: a solution to avert business failure
- Customer may define initial requirements
  - Perhaps as part of a competitive tendering process
  - Can be too abstract, high-level, imprecise, incomplete
  - Can have a biased viewpoint, make tacit assumptions
  - Can focus narrowly on one make-or-break issue
- Developer must compensate for this
  - Wider scope may deliver many additional business benefits
  - Analysis must therefore include the whole business context

Communication Issues



Customer
Understands the business domain and its jargon, but...

Compiled, total implicit expertise: cannot express systematically

Little knowledge of software systems, their limitations

Developer
Little knowledge of the business domain, or vocabulary; and...

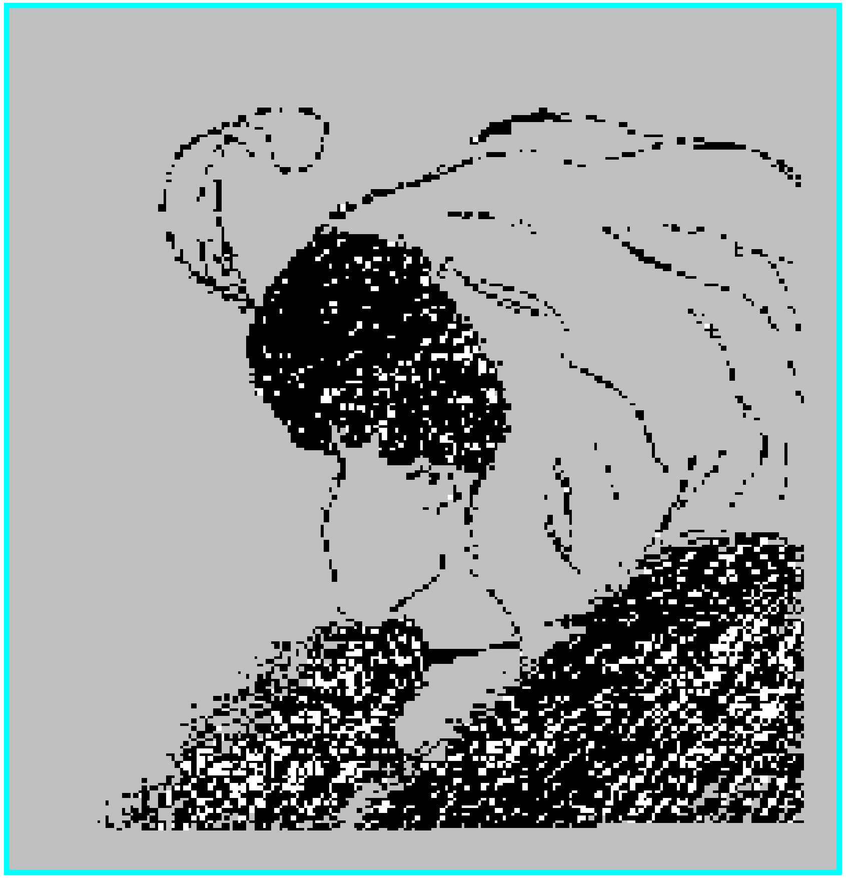
Limited, explicit model of the business, expressed systematically

Understands software systems' capabilities very well

How can we maximise this overlap?

Perception Experiment

What do you see in this image ?



Psychological Bias

- The customer has:
  - a total, but implicit (“compiled”) knowledge of the business
  - only limited ability to articulate this systematically
  - a narrow focus (“tunnel vision”) on the make-or-break criterion
- The developer has:
  - an incomplete, but explicit appreciation of the business
  - a tendency to produce over-simplified but systematic models
  - a global focus on the entire business
- How to maintain focus:
  - don’t impose structure on the customer – let him/her lead
  - use short prompts: “who does X?”, “what else does Y do?”
  - capture the breadth of the business, not a few functions in depth
  - active listening: “so you need X”, “have I understood Y?”

Useful Techniques

- Interviewing
  - stakeholders must own the result
    - let the stakeholders lead the discussion
    - use terminology from the stakeholder's domain
  - use non-directive interviewing style
    - use brief prompts only, "what", "why", "who", etc.
    - don't use forced-choice, multi-part questions
    - reflect back what is captured, for confirmation
- Workshops
  - collect multiple stakeholder viewpoints
    - exaggerate extreme viewpoints to see conflicts
  - balance, or resolve conflicts of interest
    - choose the solution which preserves balance of power

Workplace Politics

- The stakeholders
  - wield power: success or failure depends on their acceptance
  - have vested interests in how their job-roles are supported
  - new system may affect their relative status, importance
  - poor understanding, or fear of technology affect take-up
  - management can impose wrong system for political reasons
  - industrial relations can influence system adoption
- Viewpoint analysis
  - get stakeholders to express their “bluesky wishlists”
  - encourage extreme, caricatured, or selfish viewpoints
  - reveal conflicts, stimulate discussion in a humorous way
  - defuses tension, achieves resolution

Lab 2: Viewpoint Analysis

- For the ATM case study
  - imagine you are commissioning the first ATM cash dispenser
  - conflicts over availability, security of money held in banks
- Caricature extreme stakeholder positions
  - Bank Manager – responsible for security
  - Cashier – manages customer transactions
  - Customer – access to money
- How does the solution balance concerns?
  - what is the chosen solution?
  - how does it balance concerns of each stakeholder?

Run a Poll

Requirements Drift

- Requirements always change during capture
  - modelling highlights unforeseen issues
  - the business environment may change
  - the software platform may change
- Prioritize requirements
  - identify stable, vs. volatile requirements (change more often)
  - classify as essential, necessary, desirable and optional
- Agree staged delivery schedule
  - essential requirements – without which the system is pointless
  - necessary requirements – on which essential features depend
  - desirable requirements – may be dropped if time runs out
  - optional requirements – not considered unless time permits

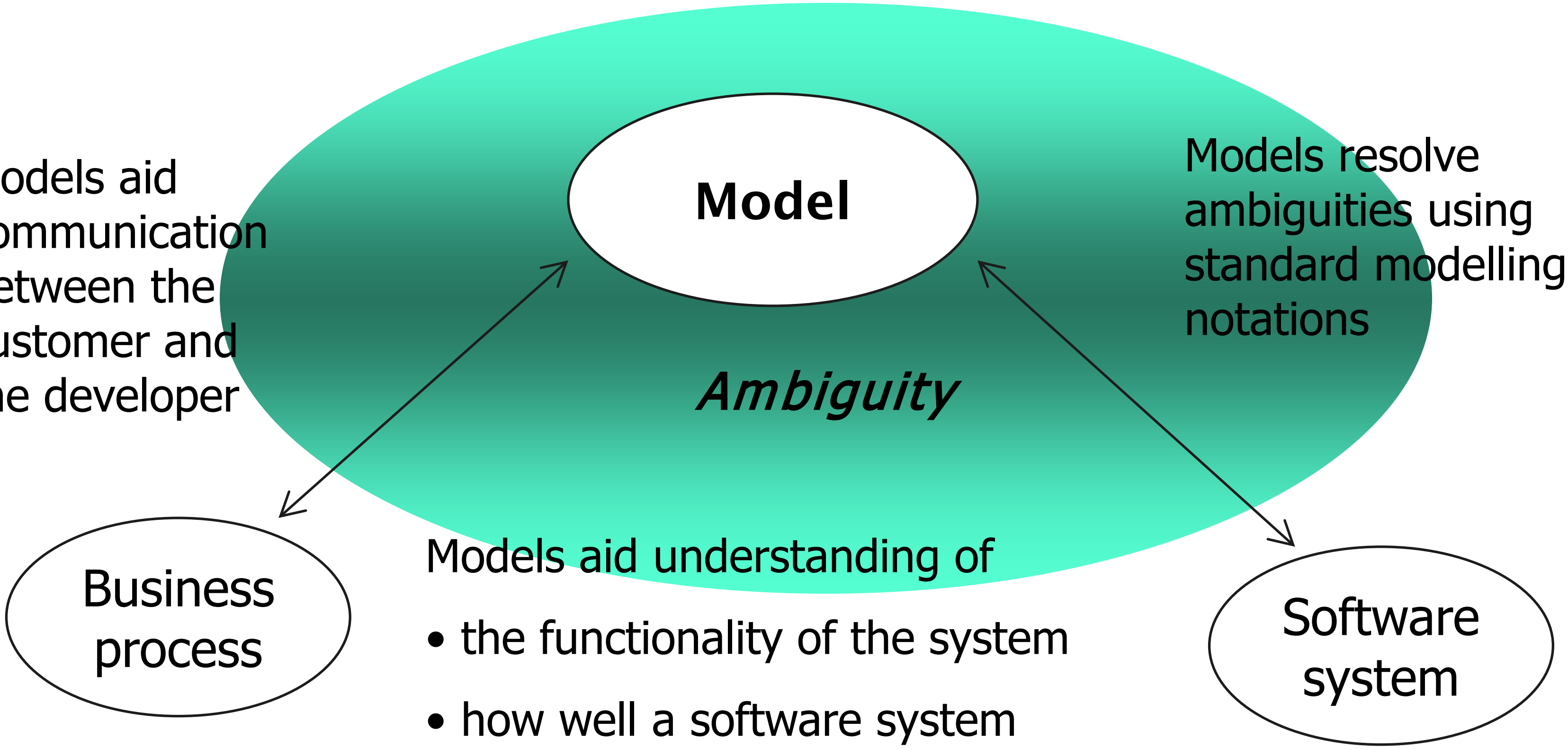
Requirements Review

- The requirements review
  - should be a regular event
  - should involve all stakeholders
- Should check for
  - Validity – proposed system meets the customer's needs
  - Consistency – no logical contradictions in requirements
  - Completeness – no gaps, or missing requirements
  - Coherence – no conflict between functional/non-functional
  - Feasibility – can deliver on time and within budget
  - Verifiability – can test system against these requirements
- Faults in requirements
  - at best, expensive to fix
  - at worst, impossible to fix

Coordination Issues

- Small Projects
  - capable of being understood by all the team
  - developers rotate easily onto different parts
  - good communication within small groups
- Large Projects
  - no one person understands the whole
  - increased importance of modelling, abstract or partial views of the system, decomposition into subsystems
  - developers handle particular modules, subsystems
  - communication good within subdivisions, but poor across the project as a whole
  - increased reliance on documentation, designs, interfaces
  - increased use of version control (eg Git), change tracking

Why Build Models?



Models aid communication between the customer and the developer

Model

Models resolve ambiguities using standard modelling notations

Ambiguity

Business process

Software system

Models aid understanding of

- the functionality of the system
- how well a software system matches the desired process

Models and Management

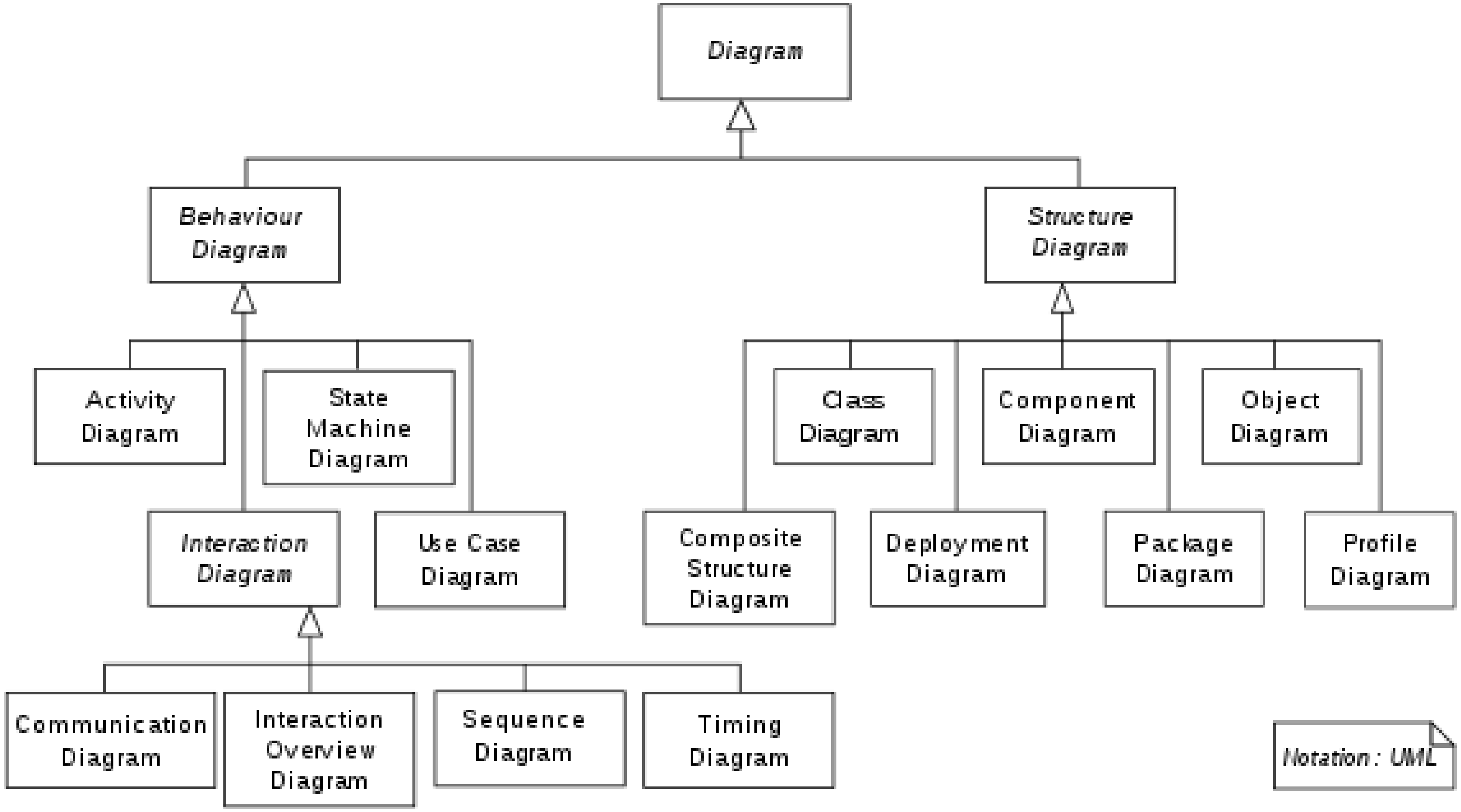
- Models help with coordination
  - abstract views of the whole system
  - detailed views of subsystems
  - models are platform-independent
- Models help with communication
  - offer a communication framework within/across teams
  - clarify and document structures and relationships
  - reveal/generate new ideas and possibilities
- Models help with the product
  - support decomposition, modular design
  - support code generation using CASE tools
  - support QA scenarios, test generation

Unified Modeling Language

- UML 2.5.1 is the standard design notation for documenting modern software systems (OMG, Dec 2017)
- UML is a precise modelling language – like circuit diagrams
  - UML syntax – expresses what diagrams are legal
  - UML semantics – expresses what diagrams mean
- UML supports different stages of the lifecycle
  - analysis models of the perceived world
  - design models documenting implementations
- UML is learned by every software engineer
  - major focus of this course – accurate usage of UML

UML Diagrams

a class diagram!



UML Diagrams – I

- Use Case Diagram
  - used in analysis, to capture functional requirements
  - used in testing, to perform system walk-throughs
- Class Diagram
  - used in analysis, to capture initial data and relationships
  - used in database design, to specify normal models
  - used in coding, to document implementation details
- Activity Diagram
  - used in analysis, to capture control flow constraints
  - used in analysis, to capture concurrent and distributed flows
  - used in design, to specify data flow, procedures, exceptions

UML Diagrams – II

- State Machine Diagram
  - used in analysis, to capture state-dependent constraints
  - used in design, to specify message accept/reject protocols
  - used in GUI design, to specify user interface behaviour
- Sequence Diagram
  - used in coding, to document method call-graphs
- Package Diagram
  - used in coding, to document software organisation
  - used in design, to specify architectural layers and pipelines
- Deployment Diagram
  - used in design, to specify distributed systems

UML 2.5 Reference

- OMG Standard
  - http://www.omg.org/spec/uml/ - OMG portal
  - dense infrastructure and superstructure docs
  - not a good read, but a technical standard
- Quick-Reference
  - M Fowler. UML Distilled, 3rd ed., Addison-Wesley, 2003.
  - D Pilone, N Pitman. UML 2.0 in a Nutshell, O'Reilly, 2005.
- Public Website
  - https://www.uml-diagrams.org/

UML 2.5 Tools

- Open Source free UML Tools
  - check whether they support UML 2.5 (currently 2.5.1)
  - best tools that build underlying logical models include: Papyrus, Modelio, Astah
  - simple text-based, markdown-style sketching tools include: nomnoml, plantuml, UMLet
- Professional paid-for UML Tools
  - No Magic, inc.: MagicDraw UML
  - Visual Paradigm: UML 2.2, free community edn., also 30-day free trial edn. http://www.visual-paradigm.com/
  - Sparx Systems Enterprise Architect: UML 2.5, 30-day free trial edn. http://www.sparxsystems.com.au/

Summary

- Project management is about managing people, the product, the process and the project, to minimize risk
- Software systems interact with people- and paper-based business systems and with legacy computer systems
- Requirements are elicited in cycles and classified into functional, non-functional and usability requirements
- The customer and developer have different mind-sets – bridge the communication gap using non-directive interviewing
- Stakeholders have different vested interests – resolve workplace conflicts through workshops and viewpoint analysis
- UML 2.5 is the standard design notation – UML models used to coordinate development throughout lifecycle