Process Models

Agile

- Embrace change – requirements never fixed
- Deliver early and deliver often

Lean

- Think big, act small, fail fast

Ethics

Australian Computer Society (ACS)

- Primary of Public Interest
- Enhancement of Quality of Life
- Honesty
- Competence
- Professional Development
- Professionalism

Requirements

Functional Requirements

- Requirements (or capabilities) for functions (specific behaviour) that must be performed by the system
- Primary focus of most requirements activities

Non-Functional Requirements

Constraints on performance or quality

- Product Properties – Requirements on the behaviour of the product (min 8 transactions per second)
- Process Properties – Requirements on the practices used to develop / produce the system (Follow a standard)

Elicitation

Interviews

Effective for understanding problem and eliciting general requirements

Workshops

Multiple stakeholders; resolve conflicting requirements, quickly gather broad system usage

Focus Groups

Broad stakeholder representation. Gather broad-based ideas

Observations

Time consuming. Users often cannot describe everything they do.

Questionnaires

Inexpensive and easily administered to remote sites. Good questionnaires difficult to write

System Interface Analysis

Look at other system's functionality

User Interface Analysis

Study existing systems. What should be replicated and avoided

Document Analysis

Existing system documentation. Industry standards or legislation

Use Cases

<<include>> relationship

Factor out common behaviour in use cases (scenario always uses included steps)

<<extend>> relationship

Factors out optional behaviour in use cases (when there are optional or uncommon steps)

Prioritisation

MoSCoW

Must have, Should have, Could have, Won't have

Review

Types of Reviews

Technical Review

Review for conformance to standards or achievement of project milestones

Software (Fagan) Inspection

Peer review with formal process. Focus on defect detection and description

Structured Walkthrough

Less formal than inspection. No formal data collection

Audit

External review of work product. Usually late in the process

Inspection Participants

Moderator

Responsible for leading inspection process

Recorder

Keeps records of all significant inspection results

Producer

Responsible for work under review

Reader

Presents work instead of producer in formal inspection

Reviewers

Directly concerned with, and aware of work under review

Inspection Process

Request → Entry → Planning → Overview (optional) → Preparation → Inspection Meeting → Rework → Follow-up → Exit → Release

Issue Classification

Major, Minor, Grammatical, Questions

Software Testing

Validation and Verification

Validation

"Are we building the right product?"

Verification

"Are we building the product right?"

Stages of Testing

Development Testing

System is tested during development to discover bugs and defects. Unit, Integration / Component, System testing

Release Testing

Separate testing team a complete version of the system before it is released to users

User Testing

Users or potential users of a system test the system in their own environment. Types of user testing:

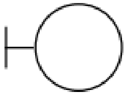
- Alpha Testing – Users of the software work with the development team to test the software at the developer's site
- Beta Testing – A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers
- Acceptance Testing – Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment

UML

Types of Classes

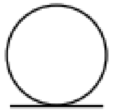
Boundary Class

Separate the interfaces from the rest of the system. Handles communication with the environment (users and other systems)



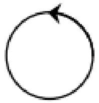
Entity Class

Functionality dealing with the storage and handling of long-lived (potential persistent) information. Often are general to many use cases



Control Class

Controls interactions between a group of objects. Functionality specific to one, or a few use cases, and not naturally placed in the other class types



Relationships between Classes

Association

Defines a semantic relationship between classes. An association has; a name, at least two association end, each end attached to a class.

Aggregation – variation of the "has a" association relationship

Composition – can exist via a part of (e.g. Carburettor "is a part of" Car)

Generalisation – defines inheritance between a super and sub class

State Machine

Moore – actions associated with states or transitions

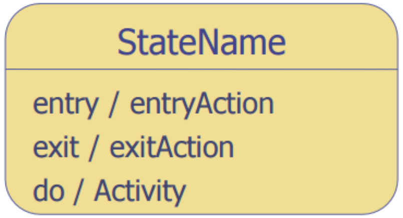
Mealy – actions may depend on system state as well as trigger

Quantitative state is the current values of all of an object's attributes

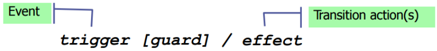
Qualitative state is the current status of an object during its lifespan

Notation

State



State Transition



Measurement

What can we measure

Products

System (size, defect, density), Module (length, percent reused, independent flow paths), Defect (type, origin, detection, severity, effort to fix)

Processes

Development (elapsed time, effort, phase containment), Testing (tests passed), Maintenance (staff hours per request)

Projects

Resources, Progress relative to schedule, Productivity, Product characteristics, Process characteristics

Structure-Related Metrics

Problem Complexity, Algorithm Complexity (time and space), Structural Complexity (control flow, data flow), Cognitive Complexity (human understanding)

Measurement Etiquette

Do

- Use common sense and organisational sensitivity when interpreting metrics data
- Provide regular feedback to the individuals and teams who have collected the data
- Work with engineers and teams to set clear goals and the metrics that will be used to measure them

Don't

- Don't use metrics to appraise individuals
- Never use metrics to threaten teams or individuals
- Don't obsess about a single metric to the exclusion of others
- Don't treat metrics data that identifies a problem as "negative"

Software Estimation

How to Estimate

Analogy

Identify similar projects and estimate based on these previous projects. Accuracy can be improved by partitions the project and estimating each part. *International Software Benchmarking Standards Group (ISBSG)*

Wide-band Delphi

Get multiple experts/stakeholders, each person give estimation anonymously.

Software Size Measures

Syntactic (e.g. Lines of Code (LOC, KLOC)), Semantic (e.g. Function Points)

Function Points

Developed by Albrecht (1979). Based on a user view of the system:

- **External Inputs** – input screen
- **External Outputs** – output screen
- **External Enquiries** – prompt for input
- **Internal Logical Files** – database table
- **External Interface Files** – database table shared between applications

Function Points $= 4\times EI + 5\times EO + 4\times EQ + 10\times ILF + 7\times EIF$

Simple Estimation Technique

Each entity class in a model will need; 1 user interface, 1 data access, half helper class. Assumes you know developer productivity (typical: 2 - 20 classes/month, average: 4 - 8)

$$E = 3.5 \times \frac{\text{Model Classes}}{\text{Productivity}}$$

COCOMO 2

Developed by Barry Boehm

Application Points

Number of separate screens displayed (simple=1, average=2, complex=3). Number of reports produced (simple=2, average=5, complex=8). Number of 3GL modules that must be developed to supplement the 4GL code (10 object points per module)

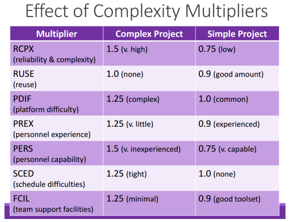
Formula

$$PM = \frac{NAP \times \left(1 - \frac{\%reuse}{100}\right)}{PROD}$$

PM – effort in person-months. NAP – number of application points. PROD – productivity (Developer Experience and Capability / CASE Maturity and Capability, PROD), (V Low, 4), (Low, 7), (Nominal, 13), (High, 25), (V High, 50)

Multipliers

RCPX – product reliability and complexity. RUSE – reuse required. PDIF – platform difficulty. PREX – personnel experience. PERS – personnel capability. SCED – required schedule. FCIL – team support facilities



Effect of Complexity Multipliers

Multiplier	Complex Project	Simple Project
RCPX (reliability & complexity)	1.5 (v. high)	0.75 (low)
RUSE (reuse)	1.0 (none)	0.9 (good amount)
PDIF (platform difficulty)	1.25 (complex)	1.0 (common)
PREX (personnel experience)	1.25 (v. little)	0.9 (experienced)
PERS (personnel capability)	1.5 (v. inexperienced)	0.75 (v. capable)
SCED (schedule difficulties)	1.25 (tight)	1.0 (none)
FCIL (team support facilities)	1.25 (minimal)	0.9 (good toolset)

Software Process Improvement

Process → People (skills, training, motivation and management) → Technology (tools, methods and techniques) → Process

UX Design and Requirements

Components of UX

- **Useful** – original and fulfils a need
- **Usable** – easy to use
- **Findable** – navigatable
- **Credible** – trust and belief
- **Accessible** – disabilities
- **Desirable** – evoke emotional

Context and Form

- **Empathize** – Explore the human context
- **Define** – Identify and select the strongest forces
- **Ideate** – Generate forms at multiple boundaries
- **Prototype** – Realize minimal form
- **Test** – Stage the ensemble and look for misfits

Written by Daniel Fitz