ICT2113 SOFTWARE REQRS: NOW & NEAR FUTURE

12. Exploring UML’s Passport to Success – Get Big Picture

Prof Kevin

Today’s agenda

1. Video – UML and why you need it.
2. Objectives.
3. Prior knowledge from previous week, and LS #4 in week 4.
4. Content – diagrams, UML strengths, notation, and index of fundamental shapes.
5. Active learning – ???.
6. Epilogue.

Today’s agenda OLD***

1. Video (Step 1)
2. Learning challenges x 3 (Step 2)
3. Prior knowledge for lesson (Step 3)
4. User story (Steps 4, 5, 6, 9)
5. Two basic prioritization schemes (Steps 4, 6, 9)
6. Postscript reflection (Step 9)
7. RE-Jeopardy (Step 9)

Video

One slide & 4:10 mins clip

Please read these instructions

Today's business challenges are daunting, especially with escalation of emerging technology and consequent new ways and means of working. Yet, in all this turmoil, software industry finally has solution to disarray in designing, communicating, and documenting its products and services under development: Unified Modeling Lanaguage (UML).

You may have seen two videos circulating on YouTube, "I hate UML" and "What went wrong with UML tools? (and why do we need another?)" but usage statistics show different picture, that UML is industry favourite.

Instead, I'm showing you pro-UML video by Károly Nyisztor @knyisztor, long time contributor of software engineering videos in his channel. He provides sober and compelling arguments about why we should embrace UML. As someone whose spend decade on learning all of UML secrets, I heartily support this video. I hope you will too. Signal when you understand, and are ready to begin.

Objectives

One slide

Today’s objectives

Be able to explain to another person …

- ❑ ... history of UML.
- ❑ ... what are UML’s strengths and why they make UML so important to software engineering.
- ❑ … different kinds of notations in UML

Draw basic distinguishing aspects of index of fundamental shapes in UML diagrams’ notations’.

Learning challenges OLD***

Two learning challenges for today are ...

1. Write two user stories for your software project.

2. Corkboards and bulletin boards have been serving comms needs of businesses for hundreds of years. Why should we see their use in User stories, Kanban, and Scrum as innovative?

[Post answers to these after class for feedback.]

Step 2

Prior knowledge

Five slides

Key[1] prior knowledge: previous LS[2]

Be able to explain to another person …

1. … UseCase2.0 usecase, story, and slice.
2. … effective writing five aspect classifications for organizing UseCase templates.

In addition, we're going to recap fundamental aspects of engineering models (LS #4).

Notes:

[1] This or present LS's Key prior knowledge is previous LS's Objectives.

[2] Previous LS is #11 Rejuvenating reqrs capture, i.e., 11th week in module.

UseCase2.0’s usecase, story, slice

In 2011 “Use-Case 2.0: The Guide to Succeeding with Use Cases”, software community introduced to UseCase2.0, new way of applying 1987 UseCases that is lightweight, lean, scalable, versatile, easy-to-use, and inspired by User stories. At UseCase2.0’s core are ‘UseCase, story, and slice’.

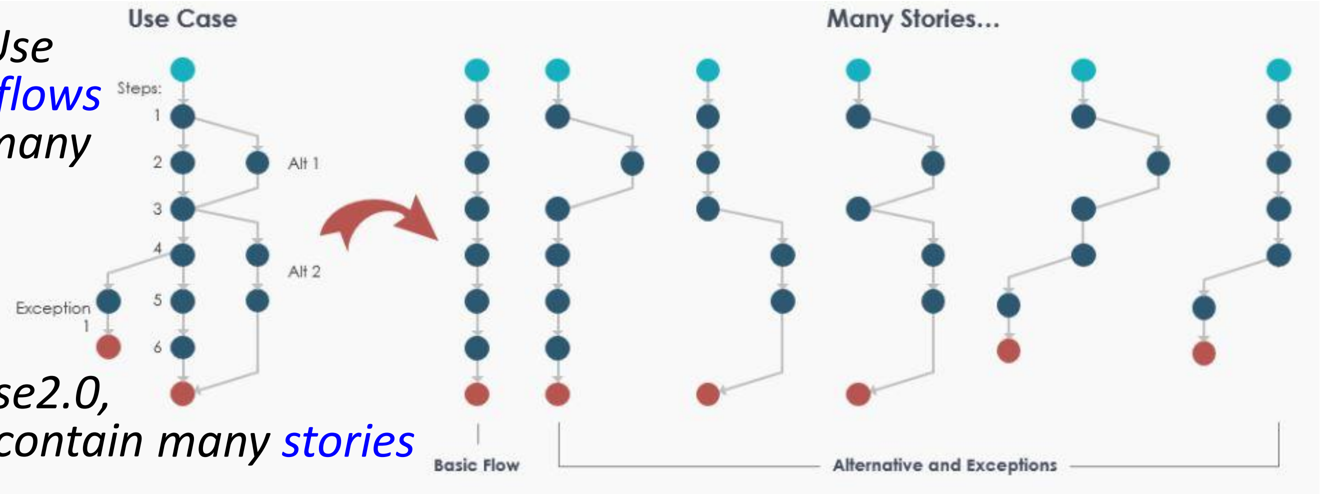
UseCase. This contains numerous steps performed by system that yield observable results of value to particular user. These steps are grouped into one or more flows. Simplest and most straightforward flow to yield result of value is called primary/basic. If certain obstacles to unbroken execution of basic flow are predicted, then UseCase will have workarounds called alternative flows.

Story. These are akin to User stories. Sure, grammar template is probably different, but we can now appreciate that using User story-like entities would enable most of Agile-like qualities that UseCase2.0 claims to have.

UseCase2.0’s usecase, story, slice (cont)

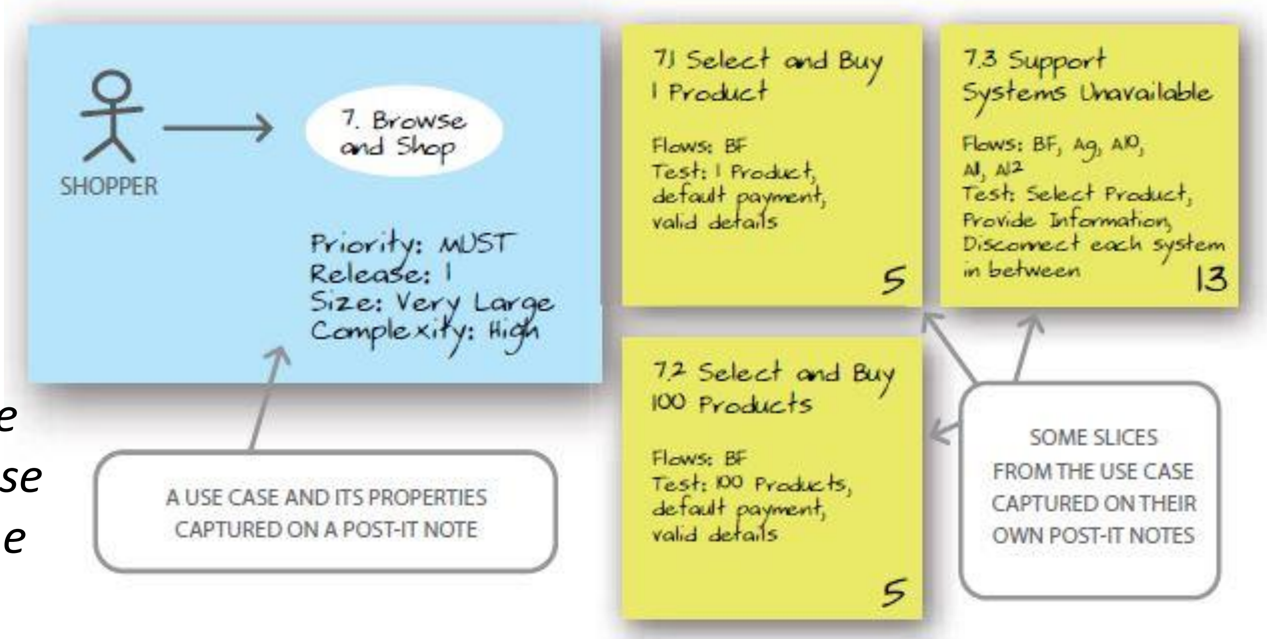
Slice. This is ... a flow in UseCase (see upper right), starting with primary, followed by additional slices for each alternate (if exists). Each slice can be developed — designed, implemented, tested — and iteratively integrated into ever-expanding kernel of finished code. What’s more, board planar artifact (see lower right) can be employed in USeCase2.0-driven SDM to continuously plan and control processing of progressive reqrs.

In 1987 Use Case, its flows contain many steps



In UseCase2.0, its slices contain many stories

Task board with sample UseCase & some slices


SHOPPER
7. Browse and Shop
Priority: MUST
Release: 1
Size: Very Large
Complexity: High
A USE CASE AND ITS PROPERTIES CAPTURED ON A POST-IT NOTE
7.1 Select and Buy 1 Product
Flows: BF
Test: 1 Product, default payment, valid details
5
7.3 Support Systems Unavailable
Flows: BF, A9, A10, A11, A12
Test: Select Product, Provide Information, Disconnect each system in between
13
7.2 Select and Buy 100 Products
Flows: BF
Test: 100 Products, default payment, valid details
5
SOME SLICES FROM THE USE CASE CAPTURED ON THEIR OWN POST-IT NOTES

Classifying UseCases templates by aspect

Template style. Alistair's effective writing style for templates is:

- one column of plain prose.
- numbered steps.
- no "if" statements.
- combinations of digits and letters, e.g., 2a, 2a1, 2a2 and so on, for sections.

Some people like to put UseCase description into table. However, lines on table create lots of visual noise, and consequently obscure actual writing.

Organizing templates. Foundation for effective writing of UseCases is templates that lay out outline sections appropriate for different content for all possible contexts. Common to all templates are actors and goals they want to achieve; after those are other aspects. They are as follows:

- (degree of) Ceremony - casual / dressed.
- (kind of) Operation - business / system.
- (level of) Scope - corporate / computer system / system innards.
- (choice of) Info hiding - white-box / black-box.
- (level of) Organization - strategic / user / subfunction.

Since Alistairs effective writing is not applied to UML diagrams, these aspect classifications are not official keywords in UML.

UseCase writer can associate their UseCase with only one aspect, or with combination of upto five aspects, e.g., 'casual business corporate white-box strategic', there are total of 72 unique possible combinations: (2 ^^ 3) * (3 ^^ 2). Of course, templates can be derived for combinations not including all five classifications.

Fundamental aspects of engr models

Model is stylized and portable representation (physical or virtual) of one or couple of aspects (either structural or behavioural) of some real usually physical entity.

Every model is developed and used for one of these purposes:

- Analysing what you know about entity.
- Uncover what has been hitherto unknown about entity.
- Manufacturing entity.
- Educating people about entity.

Every model's physical form or virtual rendering of that physical form, is in one of these.

- 1D – text & stream of tokens.
- 2D – graph, chart, waveform, photo & drawing.
- 3D – toy, display, figurine, mold & mockup.
- 4D – windtunnel, tesseract, hour-glass, light thru prism, 3D printing & electric current thru quartz crystal.

Core properties of model are:

- Notation – appearance and presentation rules, including alternatives, of constituent elements.
- Semantics – specification and categorization of constituent elements (singly or in sets/combinations); a.k.a., meaningOf.

Any questions?

Next slide

40 sec countdown

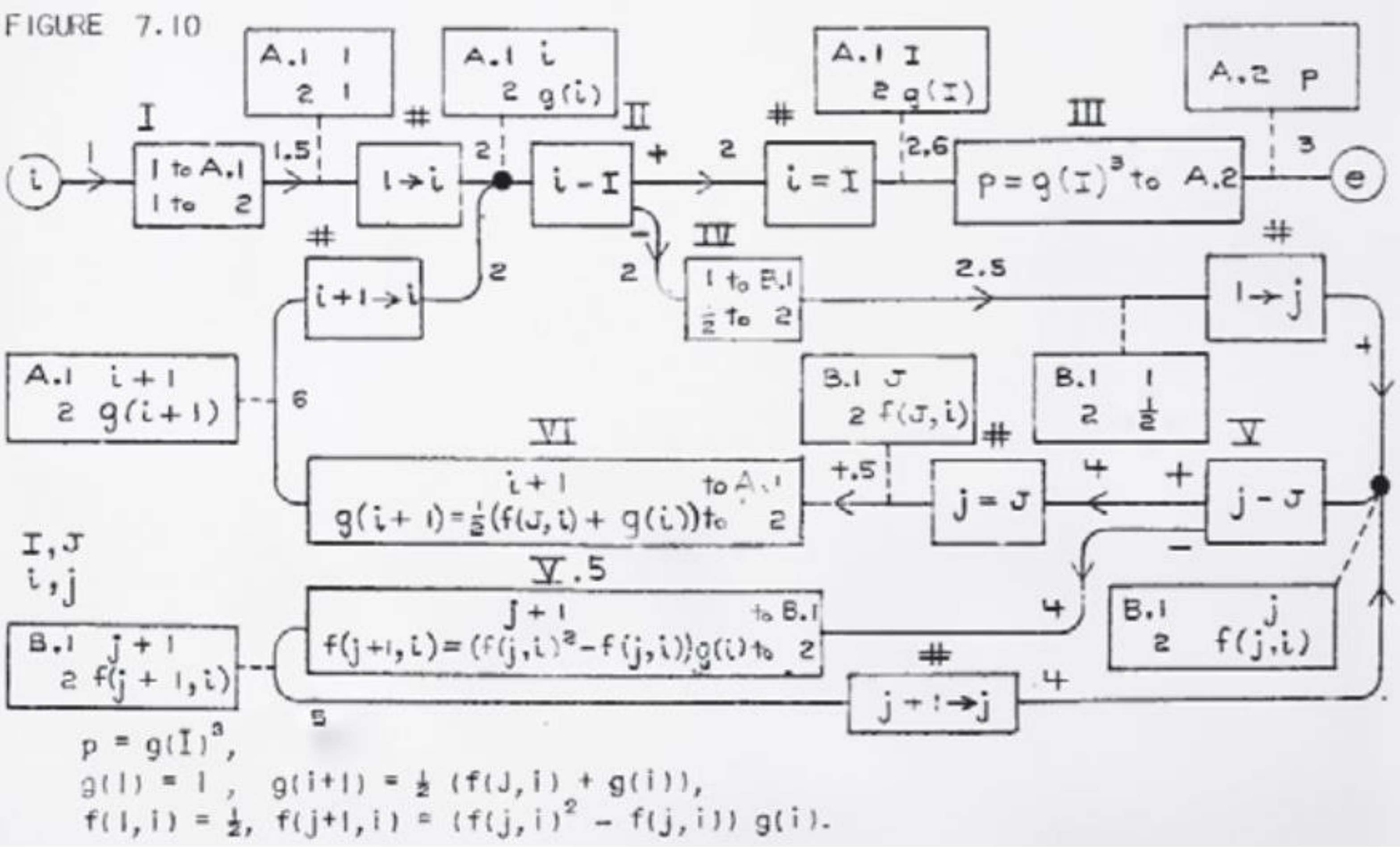
Content

Thirteen slides + two active learning tasks

Software diagrams started with flow diagrams

Diagrams with fixed conventions of material flows and process sequences were already commonplace when first electronic digital computers came on scene in 1943-5. These machines were capably programmed with external patch cables and switches; why transition to internal storable programs?

Original flow diagram from von Neumann and Goldstine's 1948 paper.



It was computing giants von Neumann and Goldstine that, by appropriating flow diagram technology, unlocked potential of portable software. In their seminal 1947 paper, "Planning and Coding of Problems for an Electronic Computing Instrument", they proposed that mathematicians design and write algorithms for program, which could be recorded into flow diagrams, that are then transformed by coder into specific machine instructions.

Rise and fall of flow diagrams

Appeal of flow diagrams was twofold. They were at higher level of abstraction than program code of day — low-level machine and assembly languages — plus they encouraged industrial discipline. Von Neumann and Goldstine's flow diagrams soon found their way into UNIVAC division at Remington Rand. They were included in teaching materials that Grace Hopper and Betty Holberton developed for programming courses conducted in April 1950. By end of 1950s, with computer industry taking off, organizations could not keep up with demand of recruiting, training, and retaining skilled programmers. Hence, belief was permeating software industry that standardized flow diagrams, now better known as flowcharts, would bring order to looming 'software crisis'.

By middle of 1960s, flowchart objects were standardized by ACM, ANSI, and ISO, plus institutionalized into national curriculums and textbooks. Flowcharting was touted as necessary and basic skill for all programmers. However, by end of 1960s, with convergence of powerful hardware, modern operating systems, and higher-level languages, flowcharting's limitations began surfacing: they were not suited to depicting nonsequential code, especially GOTO statements; programmers were drawing flowcharts after, not before, writing programs they described; and engineers could not produce flowcharts that were both detailed enough to be useful guides to development, and abstract enough to avoid becoming overly complex, unwieldy, or unreadable.

1970s alternative diagrams to flowcharts

By early 1970s, new software diagrams were filling void left from increasingly less-used flowcharts. These new diagrams were from heavyweight SDMs used, and often proprietary to, developer organizations like IBM and independent and pioneering consulting firms. IBM continued to push its standardized flowcharting, but repurposed it to show what program does rather than how programmer should build it. This is evident in its JAD methodology, where flowchart diagrams were used for developing reqrs and prototypes, but not code. Other diagrams appearing in 1970s are as follows:

IBM developed its own flowcharting template that embodied 1970 ISO and ANSI standards, for its developers to use on software projects.

- Traffic Case by Ivar Jacobson, 1970.
- Soft-Systems Method by Checkland, 1971.
- Stepwise design by Niklaus Wirth, 1971.
- Nassi–Shneiderman, 1972.
- Warnier-Orr by Warnier et al, 1974.
- Hierarchical Input Process Output (HIPO) by IBM, 1974.
- Structured Design by Yourdon & Constantine, 1975.
- Structured Design diagram by Stevens et al, 1975.
- Jackson Structured Programming (JSP) in 1975.
- Structured Analysis & Design Tech (SADT) by Ross, 1977.
- Structure Chart by Lindsey, 1977.
- System Analysis & Spec by Tom DeMarco, 1978.

Then came SSADM and its diagrams

Starting in 1980, Central Computer and Telecommunications Agency in UK government conceived and released (in following year) massive methodology with regime of software diagrams that would dominate development of heavyweight information systems up to today. Called Structured Systems Analysis and Design Method (SSADM), it combined new methods with appropriated leading SDMs of past decade, including Checkland, Constantine, DeMarco, Jackson, and Yourdon. These are its diagrams and matrices.

- Data Flow (DFD) – shows flow of data through system, showing inputs, processes, outputs, and data stores; includes Context.
- Entity-Relationship (ERD) – shows entities/objects in system and their taxonomical relationships; for database design.
- State Transition – shows entities/objects and their interactions with outside world as event-triggered state changes, a.k.a., finite state machine.
- Structure Chart – shows hierarchical structure of modules in system, showing module dependencies and data flow.
- HIPO – shows hierarchy of modules in system, showing inputs, processes, and outputs at each level.
- Decision Tree – shows decisions and range of their possible outcomes, often used in logic modeling; it is flowchart-like.
- Decision Table – shows decision logic for conditions and corresponding actions; it is matrix structure.
- Data Dictionary – shows range of details about system's data elements, including their names, types, and formats.

In past 20 years, it has gone through four major updates. In year 2000, it was renamed "Business System Development" and repackaged with 21 modules. Its diagrams and matrices continue to be taught in many institutions world-wide.

“Everybody wants to rule the world”

Since 1990, several of earlier heavyweight SDMs converted to object-oriented (OO), e.g., Yourdon & Coad method in 1988. In addition, there appeared dozens more new SDMs, some heavyweight and others lightweight, that were OO from their start, such as

- RAD by James Martin since 1991.
- DSDM since 1994.
- Scrum by Schwaber and Sutherland since 1995.
- FDD by Jeff DeLuca since 1997.
- Team Software Process by Watts Humphrey since 1998
- Rational Unified Process (RUP) by IBM since 1998
- Extreme programming by Kent Beck since 1999

Finally, there were unnoticed OO-SDMs from 1960-70s such as Simula and Smalltalk with their unique diagrams. They resurfaced in 1980s resurgence of OO, along with newer OO-SDMs, such as:

- OOD Method by Grady Booch in 1982,
- OOSA by Mellor and Schlaer in 1988.
- OO Sw Constr approach by Bertrand Meyer in 1988.
- OOD (Responsibility-driven approach) by Wirfs-Brock in 1990.
- OO Modeling Technique (OMT) by Jim Rumbaugh in 1991.
- Object-Oriented SWE (UseCase-driven) by Ivar Jacobson in 1993.

So far, 35 SDMs all with their own diagrams; undoubtedly, there are more. Obviously, appeal of diagrams — higher level of abstraction than code, and industrial discipline — continues to this day. But, how do junior entrants choose what diagrams to use? Plus, chaos of design, communication, and documentation is going from bad to worse: divergent SWE practices result in professional schisms, notation creep deteriorates active understanding, and diagram archives disintegrate as superceded notation is not maintained. What it remedy?

UML conceived to be “the way forward”

UML is comprehensive, integrated framework of 2D modeling techniques from several OO-SDM for primary application in computer engineering discipline, and brainchild of three OO pioneers, Grady Booch, Ivar Jacobson, and James Rumbaugh.

- Unified – Artifacts specified in three existing OO-SDMs — those championed by each of UML's founders — are merged into single coherent spec.
- Modeling – Three aspects: a) diagrams organized into structure and behaviour categories, b) levels of compliance; stepping through increasingly deeper and difficult concepts and techniques, i.e., not all detail at once, and c) each diagram has its own set of ModelElements, plus there are common elements shared by two or more diagrams, i.e., no unnecessary or confusing duplications.
- Language – To communicate, design, and document diagrams, but not to prescribe anything else about SDM. UML users are expected to maintain industrial discipline through proper use of language.

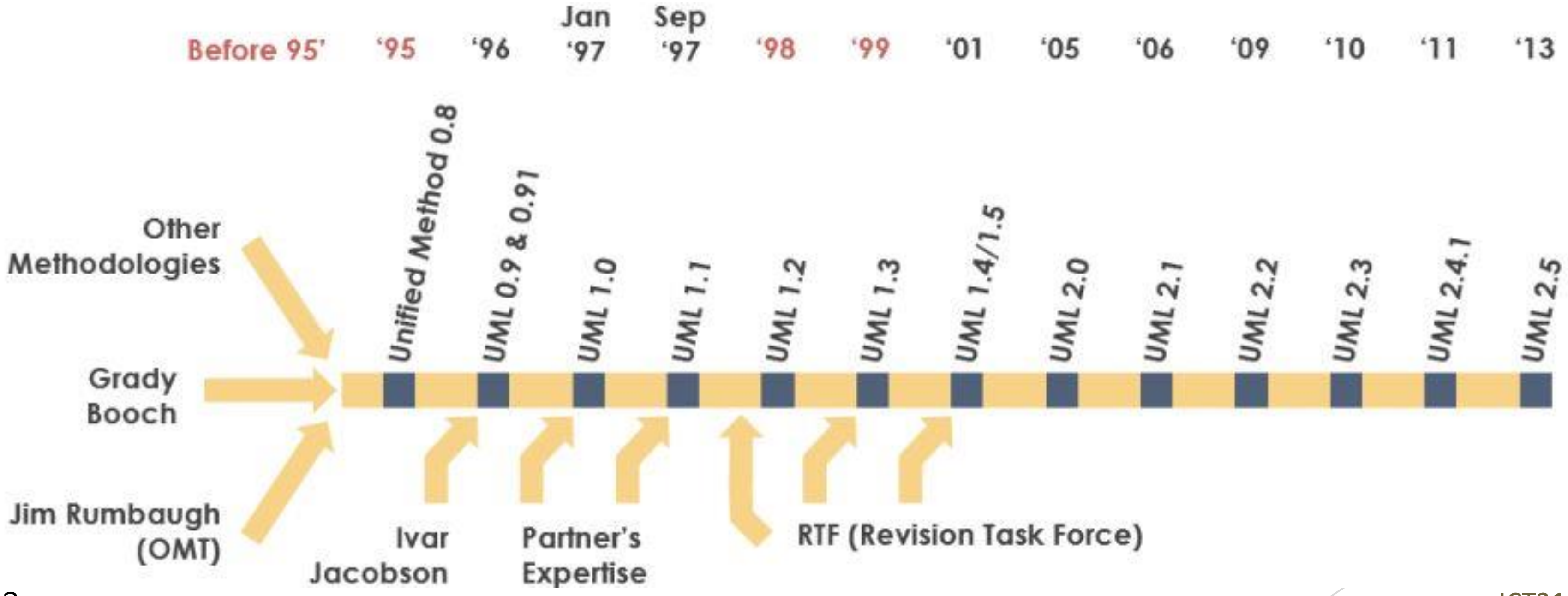
Caretaker of UML specification is Object Management Group, https://www.omg.org/. Specification consists of four volumes:

- Superstructure – framework for UML’s own model system, ie., artifacts and elements; main notation and semantic reference for casual UML users.
- Infrastructure – explains metamodel language architecture and formalism for UML; also traces UML to higher-level specifications that give UML its context in systems engineering.
- Object Constraint Language – context free language used in specification itself to specify value and constraints of features of Elements
- Diagram Interchange – protocols for interchanging with other model systems.

Superstructure up until v2.4 is organized into four parts:

1. Structure – semantics-notation for 86 * Elements.
2. Behavior – semantics-notation for 187 * Elements.
3. Supplement – 35 * auxiliary Elements for: information flow, model management, primitive Types, templates, and profiles.
4. Annexes – framing, hierarchy of diagrams, keywords, standard stereotypes, example profiles (J2EE, COM), and tabular notation.

“Three Amigos” and timeline of UML



Why we need UML

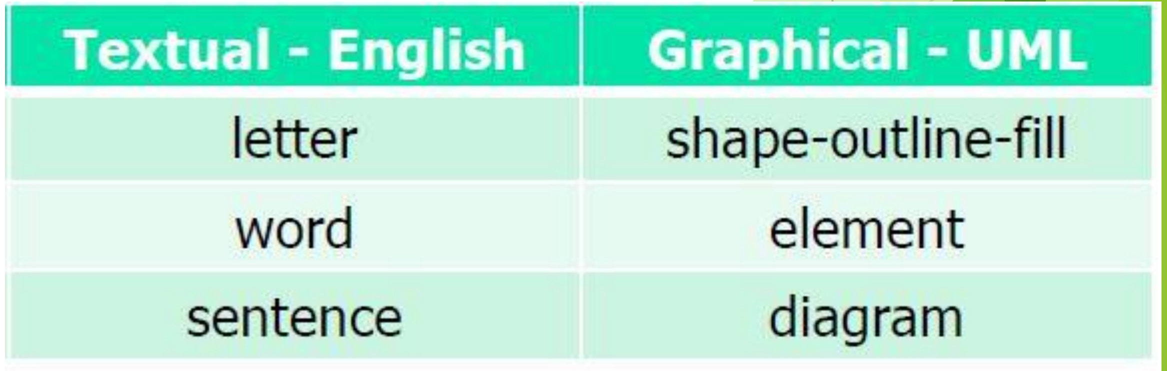
It is language, primarily graphical but augmented with stylized text comprising words and phrases.

Instead of making world conform to modeling, UML makes model conform to world, just as any language does.

It is completely independent of any SDM or industry. This means that it:

- can be applied as artifact of design, communication, and documentation for almost any function, feature, process, and industry.
- does away with most of special-purpose, single-use diagrams, i.e., no more need for SDMs to define their models. Junior entrants need only choose from inventory of 14 diagrams that can depict any feature or constituent part of any assembly or system being constructed and managed according to any SDM.

As language, it is learned in same stepwise manner — from low-granularity alphabet to very high-granularity paragraph and other sentence combinations — that is universal to our species.



Textual - English	Graphical - UML
letter	shape-outline-fill
word	element
sentence	diagram

Keep telling yourself: now learning alphabet of UML

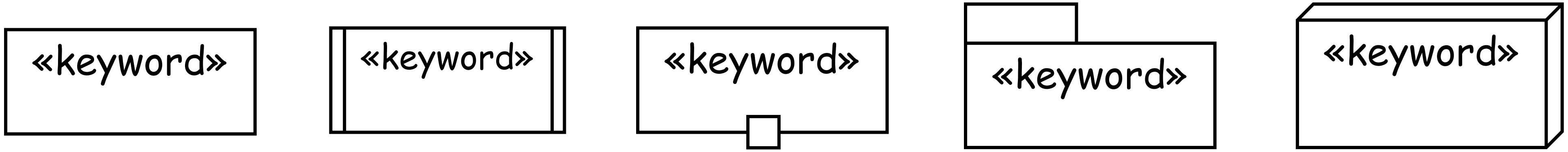
Don't look for meaning! There is none; instead, recognize patterns that facilitate recognition, e.g., square is usually structure, and roundish is typically behaviour.

UML diagrams are graphs. Therefore, each shape for each ModelElement must be understood as either vertex or edge.

What are fundamental rules about differentiating shapes?

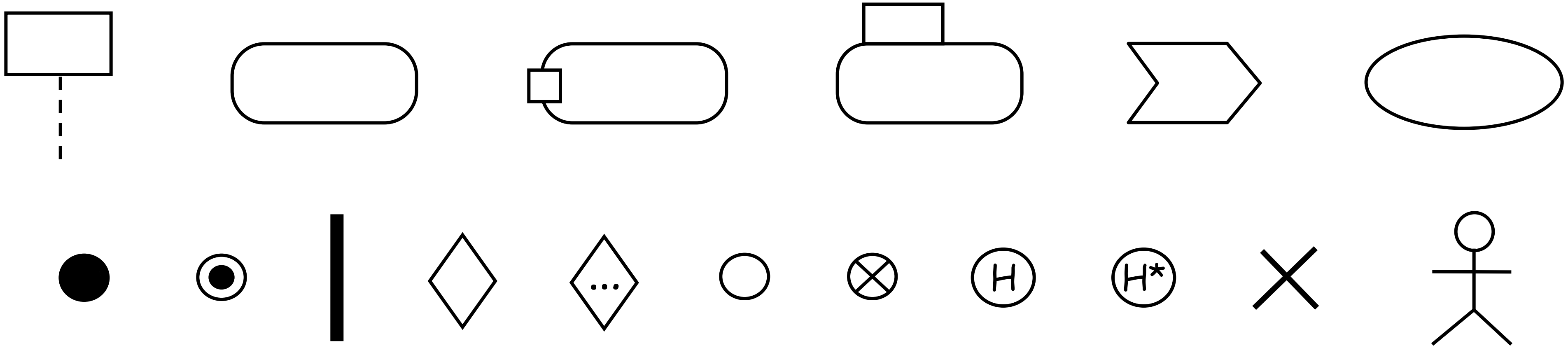
- Geometry not orientation identifies different shape.
- Glyph with solid line is a different shape from the same glyph with dashed line.
- Glyph that is filled (only black) is different shape from same glyph that is unfilled.
- Arrowhead being *open* ↑ is a different shape than from *closed* ⬆
- Size and colour cannot differentiate shape.

Vertex shape is either box or round with space inside for other Elements

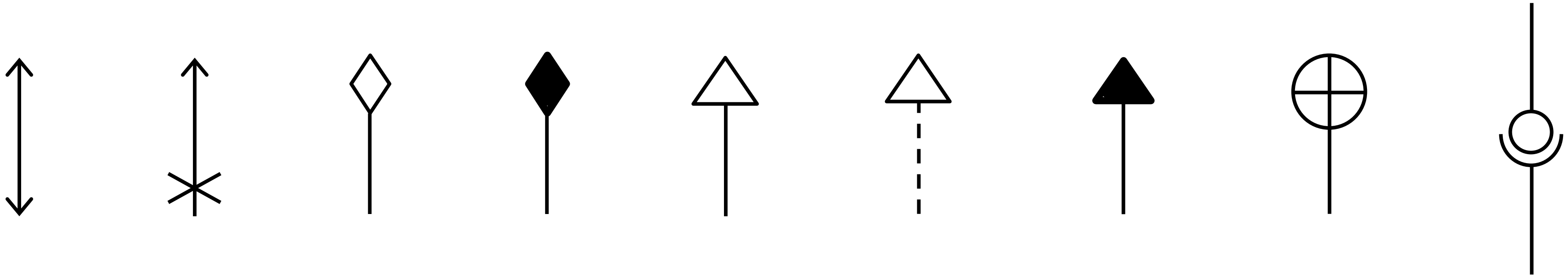


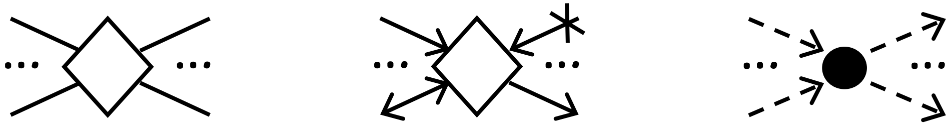
For **structure** Element

For **behavior** Element



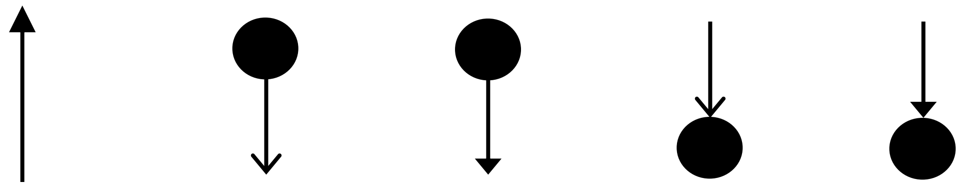
Edge shape is line with “head”; nothing inside





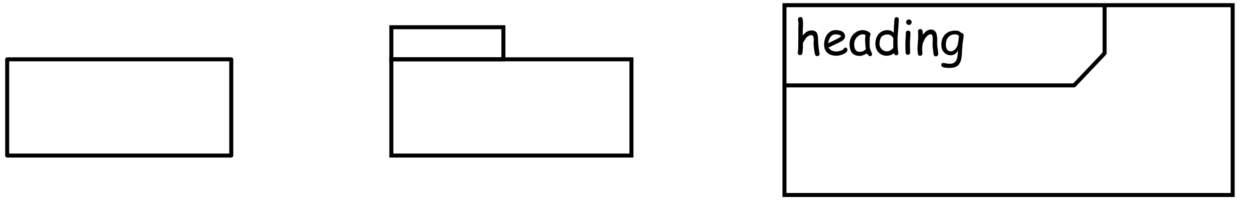
For **structure** Element

For **behavior** Element

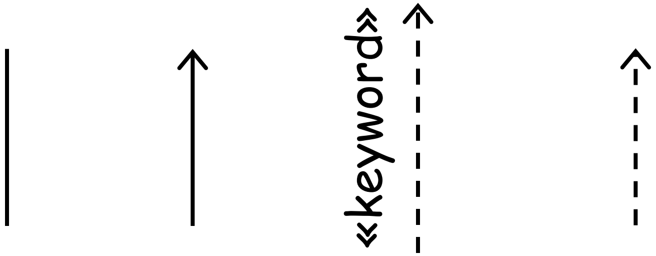


Common shapes: both edge and vertex, or used in more than one diagram

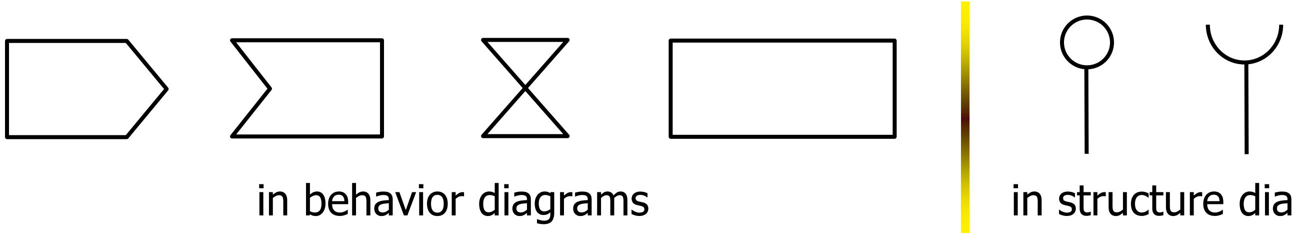
Vertex common to most structure and behavior diagrams



Edge common to most structure and behavior diagrams



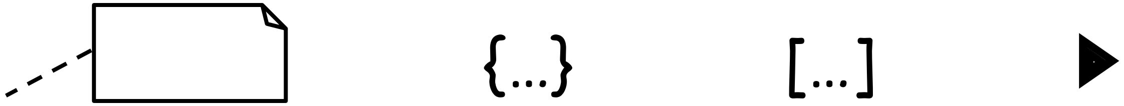
Can be either vertex or edge



in behavior diagrams

in structure diags

Used in any diagram



Text string is “anti-shape” annotation to shape

visibility operationName(dir parName: Type, …): Type {descriptor}

visibility propertyName: Type = defaultValue {modifier}

---

operatorName(operand1, operand2, …)

minValue .. maxValue

* , null

LiteralString

literalString

---

operationName(argument)

signalName(value)

Epilogue

Two slides

It is pretty clear that decades on no discipline in software development has made a legacy of tremendous duplication and lost lessons. But, it is serendipitious that UML happened when it did, at the height of chaos with our industry, and just before advent of Agile, being disruptive technology that could significantly change way of software development. UML is accepted by majority of software community because it is recognized for its importance as game changer. You should know and understand it too.

"UML is really your passport to success in software industry!"

My final words

- Contact me for …
  - regular stuff via my work email h51581@singaporetech.edu.sg.
  - more sensitive stuff, by Whatsapp 97761716 or my personal email johnmilton2000@gmail.com.
- Tomorrow's lab is 1st on subject of Innovative reqrs capture. Hopefully, your Validating was successful, and you are ready to study possible successors to RE. In preparation, ensure you are familiar with your …
  - User stories.
  - PD; yes, turn what you did earlier into a blank slate, and start a-fresh.
  - Ques concerning how to apply UseCase2.0 + effective writing.
- Keep on working through Glossary, four terms per week, and Google.
- Teachers take certain concepts being understood for granted. If you are falling behind, see me or TA Ernest for consultation.

Glossary & Google readings for next week

❑ ???

Any final thoughts?

If not …

End of learning session #12
Exploring UML's Passport to Success – Get Big Picture
by Prof Kevin

ICT2113 Software Reqrs: Now & Near Future