SECTION 4: OO METHODOLOGIES (process)

- This section is based on selected parts of the 2008 review paper “Process-Centered Review of Object Oriented Software Development Methodologies”, Ramsin&Paige

- The following slides indicate the parts of that paper that should be focussed on.

- These selected parts are discussed in outline during lectures.

- Students are expected to use private study to form a detailed understanding of these selected parts.

- Slides 6 onwards contain some extracts from the selected parts.
The paper by Ramsin & Paige is sub-divided into 9 main sections, as follows:

1. INTRODUCTION (1.5 pages)

2. BASIC DEFINITIONS (1 page)

3. OBJECT-ORIENTED METHODOLOGIES: A BRIEF HISTORY (4 pages)
   
   3.1 Seminal Methodologies: First and Second Generations
   
   3.2 The Unified Modeling Language (UML)
   
   3.3 Integrated Methodologies: Third Generation
   
   3.4 Agile Methodologies
   
   3.5 Process Patterns and Process Metamodels

_N.B.: Students should study these three sections (1, 2, 3) for CA422._
4. SEMINAL METHODOLOGIES

4.1 Schlaer-Mellor (*can omit*)
4.2 Coad-Yourdon
4.3 RDD (responsibility-driven design)
4.4 Booch
4.5 OMT (object modeling technique)
4.6 OSA (object-oriented systems analysis) (*can omit*)
4.7 OOSE (object-oriented-oriented software engineering)
4.8 BON (better or business object notation) (*can omit*)
4.9 Hodge-Mock (*can omit*)
4.10 Syntropy
4.11 Fusion

*N.B.: Students should study 7 sections (4.2, 4.3, 4.4, 4.5, 4.7, 4.10 & 4.11) for CA422.*
5. INTEGRATED METHODOLOGIES (17 page, 6 sub-sections)

NB: Apart from the short opening paragraph this section is not required for CA422

6. AGILE METHODOLOGIES (26 pages, 7 sub-sections)

NB: Apart from the short opening paragraph this section is not required for CA422

7. PROCESS PATTERNS (1 page)

NB: Apart from the short opening paragraph this section is not required for CA422

8. PROCESS METAMODELS (1 page)

NB: Apart from the short opening paragraph this section is not required for CA422

N.B.: To summarise, students are not expected to study sections 5 to 8 in any detail.
9. OBSERVATIONS AND CONCLUSIONS (4.5 pages)

9.1 Status Quo

9.2 Problem Areas

9.3 The Road Ahead

N.B.: Students should study all of the above. Specifically, key observations & conclusions of section 9 should be known and understood.
1. INTRODUCTION (1.5 pages)

“The review is carried out for the following motivations:
— to help users of methodologies … understand the processes, phase interleavings, and expected interactions that a methodology implicitly or explicitly requires;
— to provide information that may be useful for users of methodologies who wish to tailor or select processes for their particular project or context;
— to help users [evaluate] processes with respect to their [own] projects and contexts.

It is inevitable that, in such a review, issues related to modeling languages will be touched upon, but we attempt to abstract away from these details in order to focus on process; standardization efforts for the UML are of course essential and helpful here.”

2. BASIC DEFINITIONS (1 page)

“… a software development methodology … [may be] described as consisting of two main parts [OMG 2003]:
(1) a set of modeling conventions comprising a modeling language (syntax and semantics);
(2) a process, which:
— provides guidance as to the order of the activities;
— specifies what artifacts should be developed using the modeling language;
— directs the tasks of individual developers and the team as a whole; and
— offers criteria for monitoring and measuring a project’s products and activities.”
3. OBJECT-ORIENTED METHODOLOGIES: A BRIEF HISTORY (4 pages) [page 1/2]

3.1 Seminal Methodologies: First and Second Generations

“First- and second-generation methodologies are collectively referred to as seminal methodologies, in that they pioneered the unexplored field of pure object-oriented analysis and design, and in doing so laid the groundwork for further evolution.”

3.2 The Unified Modeling Language (UML)

“UML is the result of an effort to unify the visual modeling languages used in object oriented methodologies, following the realization that although they were mostly different in terms of process and life-cycle model, many object-oriented methodologies used diagrams that were in essence identical. Therefore, starting the trend of integration and unification with unifying the modeling languages seemed the logical choice. …”

3.3 Integrated Methodologies: Third Generation

“Methodologies in this category are results of integrating seminal methodologies and are characterized by their process-centered attitude towards software development, typically targeting a vast variety of applications. Integrations have resulted in huge methodologies, difficult to manage and enact … It was frustration with these methodologies that ultimately caused the agile movement … Although unwieldy and complex, integrated methodologies have a lot to offer in terms of process components, patterns, and management and measurement issues. …”
3. OBJECT-ORIENTED METHODOLOGIES: A BRIEF HISTORY (4 pages) \[\textit{page 2/2}\]

... 

3.4 Agile Methodologies

“The once common perception that agile methodologies are nothing but controlled code-and-fix approaches, with little or no sign of a clear-cut process, is only true of a small—albeit influential—minority of these methodologies, which are essentially based on practices of program design, coding, and testing that are believed to enhance software development flexibility and productivity. Most agile methodologies incorporate explicit processes, although striving to keep them as lightweight as possible.”

3.5 Process Patterns and Process Metamodels

“The advent of UML has allowed methodologists to focus on processes instead of concerning themselves with devising new modeling languages, and the experience gained from the relatively long and adventurous history of OOSDMs [object-oriented software development methodologies] has helped methodologists identify patterns and generalities among processes. Process patterns are the results of applying abstraction to process components, thereby presenting ways for developing methodologies through composition of appropriate pattern instances … Process metamodels, on the other hand, are the results of applying abstraction on the overall process …”
4. SEMINAL METHODOLOGIES

4.1 Schlaer-Mellor (can omit)

4.2 Coad-Yourdon

Fig. 4. The Coad-Yourdon model for software development—adapted from Coad and Nicola [1993].
4. SEMINAL METHODOLOGIES [Page 2/7]

4.3 RDD (responsibility-driven design)

Fig. 5. The RDD process—adapted from Wirfs-Brock et al. [1990].
4. SEMINAL METHODOLOGIES [Page 3/7]

4.4 Booch

Fig. 6. The Macro Process of the Booch methodology—adapted from Booch [1994].

Fig. 7. The Micro Process of the Booch Methodology—adapted from Booch [1994].
4. SEMINAL METHODOLOGIES [Page 4/7]

4.5 OMT (object modeling technique)

![Diagram of OMT process]

**Fig. 8.** The OMT process and its deliverables—adapted from Derr [1995].
4. SEMINAL METHODOLOGIES [Page 5/7]

4.6 OSA (object-oriented systems analysis) (can omit)

4.7 OOSE (object-oriented-oriented software engineering)

Fig. 9. The OOSE process and the models produced—adapted from Jacobson et al. [1992].
4. SEMINAL METHODOLOGIES [Page 6/7]

4.8 BON (better or business object notation) (*can omit*)

4.9 Hodge-Mock (*can omit*)

4.10 Syntropy

---

Fig. 12. Models, views and their interdependencies, showing the implicit syntropy process—adapted from Cook and Daniels [1994].
4. SEMINAL METHODOLOGIES

4.11 Fusion

Fig. 13. The Fusion process and its deliverables—adapted from Lano et al. [2000].
9. OBSERVATIONS AND CONCLUSIONS (4.5 pages) [Page 1/2]

9.1 Status Quo

“Attempts at integration, unification, and standardization have actually aggravated the problems of complexity and inconsistency … The integrated, heavyweight methodologies are very complex, and some of their [agile] competitors are little more than controlled code-and-fix methods based on good programming practices. …

The evolution process seems to have gone astray, and as a result, we are witnessing the comeback of some of the old methodologies …

The course of events suggests that any effort aimed at enhancing OO methodologies should also consider the abundant capabilities of older methodologies, neglected during the integration euphoria. …”

9.2 Problem Areas

“(1) Requirements engineering is still the weak link, and requirements traceability is rarely supported; requirements are either not adequately captured or partially lost or corrupted during the development process …

(2) Model inconsistency is a dire problem. UML has exacerbated the situation instead of improving it …

(3) Integrated methodologies are too complex to be effectively mastered, configured, and enacted …

(4) Despite remarkable achievements, agile methodologies are still not mature enough … some of the more commonly cited problems [are]… unrealistic assumptions …lack of scalability … lack of a specific, unambiguous process.

(5) Seamless development, pioneered by seminal methodologies, is not adequately appreciated and supported in modern-day methodologies …”
9. OBSERVATIONS AND CONCLUSIONS (4.5 pages) [Page 2/2]

9.3 The Road Ahead

“There is motivation for developing methodologies that use the lessons learned from UML and the long history of object-oriented methodologies in setting up a framework for software development that addresses the problem issues. The following can be pointed out as general characteristics of such methodologies:

(1) **compactness**: an extensible core is preferable to a customizable monstrosity or a generic framework with complex parameters and/or prohibitively numerous parameter options;

(2) **extensibility**: with extension mechanisms and guidelines clearly defined;

(3) **traceability to requirements**: all the artifacts should be traceable to the requirements;

(4) **consistency**: artifacts produced should not be allowed to contradict each other; alternatively, there should be mechanisms for detecting inconsistencies;

(5) **testability of the artifacts from the start**: this will allow tools to be developed to verify and validate the artifacts;

(6) **tangibility of the artifacts**: producing executable material as early as possible in the lifecycle;

(7) **visible rationality**: there should be evident rationality behind every task and the order in which the tasks are performed, and undeniable use for every artifact produced; the developers should be able to to see this logic, truly sensing that any digression will put their objectives at risk.”