A Pragmatic Documents Standard for an Experience Library: Roles, Documents, Contents and Structure

Reidar Conradi (scribe), NTNU, p.t. FC-MD/ Univ. Maryland
Victor R. Basili, Jeff Carver, Forrest Shull, Guilherme H. Travassos, FC-MD/UMD

Univ. Maryland (UMD), Version 1.54

Abstract:
This report constitutes a documents standard. It describes the major roles, processes, and documents necessary for designing, running, analyzing, synthesizing, recording and disseminating information from controlled experiments. (Other kinds of empirical studies, such as case studies, are not covered.) This description is the result of the authors' experience with replication of controlled experiments, and addresses what was learned about the needs of both the original experimenter and the replicator.

To disseminate the information needed for an experimental replication, this document proposes a web-based and thus distributed Experience Library. The corresponding document standard has been designed to make relevant documents and empirical material into a "living" experience base. To this goal, this document describes pragmatic information structures to store, disseminate, comment upon, and update documents on designing and running experiments. We describe how special care must be taken to document otherwise unwritten goals, rationales, and assumptions behind experiments and thus to support more reasoned changes (adaptations and improvements) on replications.

An accompanying set of Quality Assurance Rules regulates the evolution and use of this Experience Library. A main web portal is also described which ties together all local Experience Libraries, Document Standards, and Quality Assurance Rules.

TABLE OF CONTENTS:

1. Overall survey and concerns
2. Processes and roles
   2.1 Master Librarian
   2.2 Steering Board
   2.3 Theme Librarian (local)
   2.4 General reader
   2.5 Replicator
   2.6 Experimental designer
   2.7 Aggregator
   2.8 Subject
3. Document types
   3.1 Introduction and overview
   3.2 Software Artifact
   3.3 Experimental Instruction for subjects (Assignment Description)
   3.4 Internal Instructions for researchers (Lab Manual)
   3.5 Report and data form
   3.6 Experimental raw results
   3.7 Experimental refined results (analysis)
   3.8 Aggregated experiments across a family of replicated experiments (synthesis)
   3.9 Thematic area, covering families of studies (both general and technical)
   3.10 Course/training notes and guidelines
   3.11 Experimental package (a composite)
   3.12 List of cooperating partners
   3.13 Glossary
4. Organization of files/documents
   4.1 General document organization
   4.2 Portals
   4.3 File naming conventions
5. Intellectual (Legal) Rights
6. Examples of experiments to be included

7. Appendix A: Jeanice Singer: APA Style Guidelines for reporting experiments
8. Appendix B: OORT Lab Manual at UMD, course 735, fall 1999
9. Appendix C: OORT Lab Manual at NTNU, course 78038, spring 2000
10. Appendix D: OORT Assignment Description at UMD, course 735, fall 1999
11. Appendix E: OORT Assignment Description at NTNU, course 78038, spring 2000
12. Appendix F: The Original and the Refined OORT-4 Class Diagram x Class Description
13. Appendix G: Comments from Designing an OORT Experiment at NTNU

1. Overall Survey and Concerns

This document constitutes a **Documents Standard**. It describes the major roles (processes) and documents for designing, running, analyzing, synthesizing, recording and disseminating information from controlled experiments. This information will reside in a web-based and thus distributed **Experience Library.** Other kinds of empirical studies, such as case studies, are not covered.

That is, we should make relevant documents and empirical material into a "living" experience base. This means establishing pragmatical information structures to store, disseminate, comment upon, and update documents on designing and running experiments. Special care must be taken to document otherwise unwritten goals, rationale, and assumptions behind experiments and thus to support more reasoned changes (adaptations, improvements). An accompanying set of **Quality Assurance Rules** will regulate the evolution and use of this Experience Library. The Documents Standards and the Quality Assurance Rules will themselves reside at the **main web portal** over all local Experience Libraries.

The major roles are described in Section 2, partly following Rational's recommended set-up for use cases.

Section 3 describes the main document types, their contents and how they can be aggregated into document clusters.

Section 4 describes the more concrete document structure, including simple versioning. Some of the documents will be so-called **umbrella documents** over a cluster of related documents, e.g. for thematic areas (3.9). Each document cluster will have a shared **log of comments** with questions and answers and with documentation of relevant changes with justification. The comments are made by other readers or by the document responsible. Some of the individual document types in Section 3 may also have a similar log. The section also proposes **web portals** (4.2) and file name conventions (4.3).

Section 5 describes intellectual (legal) rights, and section 6 gives some examples.

More detailed examples are given in appendices B-G.

Note 1: The described documents in Section 3 could be implemented as hypertext documents in standard html (the proposed solution), or in a more formal Experience Management System (EMS), e.g. as the one being implemented at FC-MD. However, an html-solution does not offer built-in search facilities, aside from standard tools like AltaVista. However, the proposed EMS will make distributed document sharing more difficult, as we have to locally install a special client program for the EMS.

Note 2: Guilherme has raised the issue of **regional managers**, as a middle level between the Steering Board and the Theme Coordinators, especially to deal with cultural/language issues. I have only proposed some more informal, **deputy theme coordinators**, in order to keep the overall structure simple in the start. There is little purpose in designing an enormous infrastructure, before we have some amount of in-the-small experience with filling up and using

the proposed library. We must also expect that (possible future) co-workers will take copies of available material, without seeking consent or advice from the authors or announced coordinators, and without the resulting, "private" libraries being made publicly available and perhaps with marginal quality assurance. Our initial emphasis should be on providing a guidance service, setting up contacts with "experts" around certain themes --- possibly across country, cultural and language borders -- and then let the community grow incrementally on its own merit. The spirit of the free software community should prevail: you are free to use and modify source versions, but eventually your changes should be brought back to the public domain.

2. Processes and Roles

We will here sketch the roles to create, access, disseminate and maintain the above documents in an Experience Library. Each role is shortly described with its main activities.
Note the similarity in with roles around an experience management system (EMS), i.e. in an Experience Factory. Simple reader/writer (Consumer/Producer) roles are not covered.

2.1 Master Librarian (central)

Goal: To establish and maintain the Documents Standard (this document), ribbon web structures, and document templates (the latter two is p.t. not elaborated).

Comment: The actual filling up and qualification of a decentralized Experience Library is delegated to local theme librarians (2.3). The Master Librarian may have a web assistant or secretary, not discussed further. He is responsible to the Steering Board (2.2).

Human Actor: A software engineering researcher or professional.

Pre-condition: Trained in software engineering and experimental methods, insight in web technology.

Post-condition: The ribbon web structure for the Experience Library and the document templates must be in a consistent state, and the corresponding Documents Standard must be up-to-date. All this is guided by **Quality Assurance Rules**, applying within and across sites.

Document access: Maintains the Documents Standard itself, the ribbon web structures and the document templates. Must access the Quality Assurance Rules (although this role may be delegated to technically maintain these).

Actions:

- Establishes and maintains the **Documents Standard**, specifying typing with contents/attributes (section 3), overall document organization in a Experience Library (4), and surrounding processes and roles (2).
- Will establish a **ribbon web structure** for a decentralized Experience Library and with **web templates** for different types of documents.
- Maintains the main entry (**web portal)** of the Experience Library, with tabular information about themes, local theme librarians and their theme coordinator librarians (2.3), the Steering Board (2.2), and the Master Librarian (2.1) himself.

Ex. Some responsible person at UMD or IESE?

2.2 Steering Board (central, but distributed)

Goal: The overall objective is to **build a community** of experimental software researchers and professionals, cf. ISERN. The detailed goals are to accept new cites and research themes, and associated theme librarians. Further to establish a set of Quality Assurance Rules that regulate the maintenance and practical use of the Experience Library, and to guide and review the Master Librarian.

Comment: Monitors the overall quality of the experimental material and the theme librarians, formulates the Quality Assurance Rules, and supervises the evolution of the Documents Standard. Some of its functions fall into the realm of a **Change Control Board**.

Human Actor: A set of senior software engineering researchers, often selected among the theme coordinator librarians. One of these acts as the **secretary** of the board, routing requests and documents to and from the board.

Pre-condition: Well trained in software engineering and empirical methods, and in how the cooperation on experimental material should work.

Post-condition: That the research cooperation among the partners work satisfactorily, and that the document/web structure is kept consistent and with contents of high quality.

Document access: Overall information about proposed partners and themes, the Documents Standard, and the Quality Assurance Rules.

Actions:

- Will set up **Quality Assurance Rules** for document updating and reviewing, to be observed by the Master Librarian (2.1) and by local theme librarians (2.3). The Master Librarian may be delegated to technically maintain these rules.
- Reviews and accepts new or revised document standards, after inputs from the Master Librarian.
- Reviews and considers new partners and themes, after requests to/from the secretary.

The board will decide which of the local theme librarians should be the "coordinator" and may provide local deputies, and providing contacts to relevant co-workers.

Ex. Senior partners at UMD, FC-MD, U. Kaiserslautern, IESE etc.
In the start it may consist of the coordinator theme librarians and some others.

2.3 Theme librarian (local knowledge manager/packager)

Goal: To be responsible for local documents about a theme and a related family of experiments, and partly to certify new documents for the Experience Library from other cites.

Comment: Does the actual filling up of documents: data entry, classification/description and qualification. Will cooperate with similar theme librarians at other places, and one such is the **theme coordinator librarian**, in charge of giving advice and providing quality assurance for the actual theme. In some countries, he may have a **local deputy** to offer a more accessible person for researchers to work and consult with. A librarian may also have a local web assistant or secretary. The local, natural language may be used for some or most of documents, although we should strive to use English for central documents whenever possible (see 3).

Human Actor: A software engineering researcher, which is a Replicator (2.5) or an Experimental Designer (2.6). The latter is often the coordinator.

Pre-condition: Must be well trained and acquainted with the actual theme and with experimental methods, and with the actual structure of experimental material.

Post-condition: Keeps the local material in order and with links to related material.

Document access: Has access to most documents, e.g. a single thematic area or a family of similar or repeated experiments. The coordinator must have pre-access to new documents during a certification process, where the main documents must be available in English.

Actions:

- Maintains experimental material according to the given Documents Standard.
- The material is first put into a local work area, and then upgraded to the public area, with proper links to related material.
- Such an upgrade can only happen after pre-certification by the actual theme coordinator librarian.

Ex. colleagues at IESE, at ISERN, the Norwegian NAWUS members.

GHT: Are regional managers also needed? Must also practice a low threshold for allowing new participants, i.e. to allow local/regional sharing of experimental information before such is accessible as "knowledge" to the wider audience -- the world is more than ISERN!??

RC: Maybe, but let us start out with a shallow hierarchy.

2.4 General Reader

Goal: To learn or to be updated on experimental techniques or on a certain experiment.

Comment: Typically performed via web. Has no obligations now, but may be a cooperating colleague later.

Human Actor: Any colleague world-wide ("web cruiser" etc.).

Pre-condition: Some competence and interest around software engineering and experimental methods.

Post-condition: Perhaps initiatives to acquire more information, contact existing experimenters and get started to replicated an experiment (2.5).

Document access: Has read access to general documents such as those in Thematic Areas (2.9), but may be monitored. Must have write access to comment logs.

Actions:
- Browses, searches and reads through parts of the publicly available material.
- Produces log of comments and questions.

Ex.: colleagues at ISERN and other researchers at FC-MD, UMD, IESE etc.

2.5 Replicator

Goal: To prepare and run a replicated experiment, using previous material.

Comment: This role is a simplified Experimental Designer (2.6). He should cooperate with the original designer of an experiment, e.g. to ensure experimental quality and joint publications afterwards. This role may be placed remotely, and may
partly be delegated to research assistants.

Human Actor: A software engineering researcher, perhaps a professional.

Pre-condition: Must be trained in experimental methods (hopefully) and software engineering.

Post-condition: A complete and filled-up experimental package (3.11).

Document access: Has read/copy-modify access to the relevant information for a given experiment (documents 3.3, 3.4, parts of 3.11), and all comments, changes and results are communicated back to the original team.

Actions:

- Must negotiate a cooperation agreement with the team behind the original experiment.
- Must produce a revised lab manual (3.4) and assignment description (3.3), and reuse/change related documents. E.g. reuses corresponding artifacts (3.2) more or less verbatim.
- Must, however, relate to a revised experimental context and (re)discuss validities.
- All changes must be justified (also the syntactic and administrative ones), and well documented.
- Run the experiment, and possibly adjust the plan/assignment.
- Does initial data analysis to understand how the experiment went.
- Will write feedback comments based on how the experiment was planned and run, and on its results.
- Will later write papers/reports based upon this (cooperate with others on this), and perhaps contribute to making revised and more general models.

Ex. colleagues at IESE, at ISERN, the Norwegian NAWUS members etc.

2.6 Experimental designer

Goal: To build and run a new experiment, e.g. on testing, OORT, estimation etc.

Comment: Done as part of a course or in a small industrial environment. Will cooperate with the Aggregator (2.7) and the Replicator (2.5).

Human Actor: A software engineering researcher.

Pre-condition: Must be very well trained in experimental methods and software engineering.

Post-condition: Experimental package (3.11) "in good order", and initial papers written.

Document access: Maintaining a consistent set of experimental material (aggregated in 3.11) -- especially Assignment description (3.3), Lab manual (3.4), Report forms (3.5), Raw Experimental data (3.6) and partly Initial data analysis (3.7).

Actions:

- Will set up relevant research goals, rationale and hypotheses, partly based on insights from previously published material (3.8 and 3.9). Especially the hypotheses and local variables must be well documented and analyzed, in order to come up with a realistic experimental design.
- May look for reusable documents (e.g. software artifacts) to get started.
- Should follow a well-defined process to make a corresponding lab manual/package (3.11), consisting of a lab manual (3.4), assignment description (3.3) with extra guidelines (3.10), software artifacts (3.2), and report forms (3.5). See later on these documents.
- Must do some pre-analysis of subjects and related planning to form grouping etc.

- Run the experiment, and possibly adjust the plan/assignment.
- Must log experiences during planning and running of such a new experiment.
- Does some data analysis etc., makes new/revised models and writes up papers and reports based upon this.
- Should cooperate with Aggregator (2.7) on how to represent a family of possibly replicated experiments.

Ex. OORT and PBR researchers at FC-MD/UMD, similar at IESE, NTNU etc.

2.7 Aggregator

Goal: To combine results from a family of replicated or similar experiments, and to maintain and improve the associated experimental material for such a family.

Comment: One "responsible" experimenter will collect and process comments and results from own and repeated experiments. Cooperates with original Experimental Designer (2.6) and surrounding team (often a part of this team) and the Replicators (2.5).

Human Actor: A senior researcher.

Pre-condition: Must be very well trained in experimental methods and the actual theme.

Post-condition: Synthesized models (3.9), in form of papers and reports.

Document access: Looks at a series of experimental packages (3.11), in order to modify relevant documents accordingly (3.8 and later 3.9).

Actions:
- Will look at and analyze a series of results (3.6-3.7 as part of 3.11), in order to synthesize revised/new models (3.8) and later as part of thematic areas (3.9) and to improve experimental packages (3.11 -- especially 3.3 and 3.4).

GHT: Maybe we need regional managers/responsibles for this?

Ex. A researcher from the original team behind a family of experiments, e.g. as part of the OORT or PBR team at Univ. of Maryland.

2.8 Subject (remove later??)

Goal: Perform a planned experiment, learn or perfect a new technique.

Comment: Is the individual performer of the experiment, may be grouped. -- May drop this?

Human Actor: Student or industrial developer.

Pre-condition: Subjects must not "see" experimental material before an experiment, must be taught and coached in background material (theory, guidelines).

Post-condition: Has followed the prescribed procedure, and has filled-in the required output documents.

Document access: Reads the Assignment description (3.3) and related documents (copies for originals), fills in report forms and modifies/produces software (parts of 3.5).

Actions:
- Will be checked for background, e.g. though questionnaire, before experimental allocation (grouping, techniques, artifacts) is made.
- Prepare for the given assignment, reading the assignment description (3.4).
- Do the experiment.
- Report from it.

Ex. Students at UMD, NTNU etc., industrial developers at NASA etc.

3. Document types

3.1 Introduction and overview

All these type "descriptions" are put in one (i.e. *this*) file, thus being a **schema** for all documents. Each document type serve as a **template** for what a document of this type should contain, e.g. a lab manual (3.4).
Ex. choice of natural language: English (prime) + German, Italian, Portugese, Norwegian, …

Survey of **document types**, commented in below subsections (subsections 3.x below):
3.2 Software Artifact
3.3 Experimental Instruction for subjects (assignment descriptions)
3.4 Internal Instruction for researchers (lab manual)
3.5 Report form: defect/experiment forms, questionnaires etc.
3.6 Experimental raw results: filled-in report forms etc.
3.7 Experimental refined results (analysis)
3.8 Aggregated results/models from families of replicated experiments (synthesis)
3.9 Thematic area (both experimental methods and software engineering techniques)
3.10 Course/training notes and guidelines
3.11 Experimental package: covers all material in an experiment (documents 3.2, 3.4, 3.6, 3.7)
3.12 List of cooperation partners: university, industry
3.13 Glossary

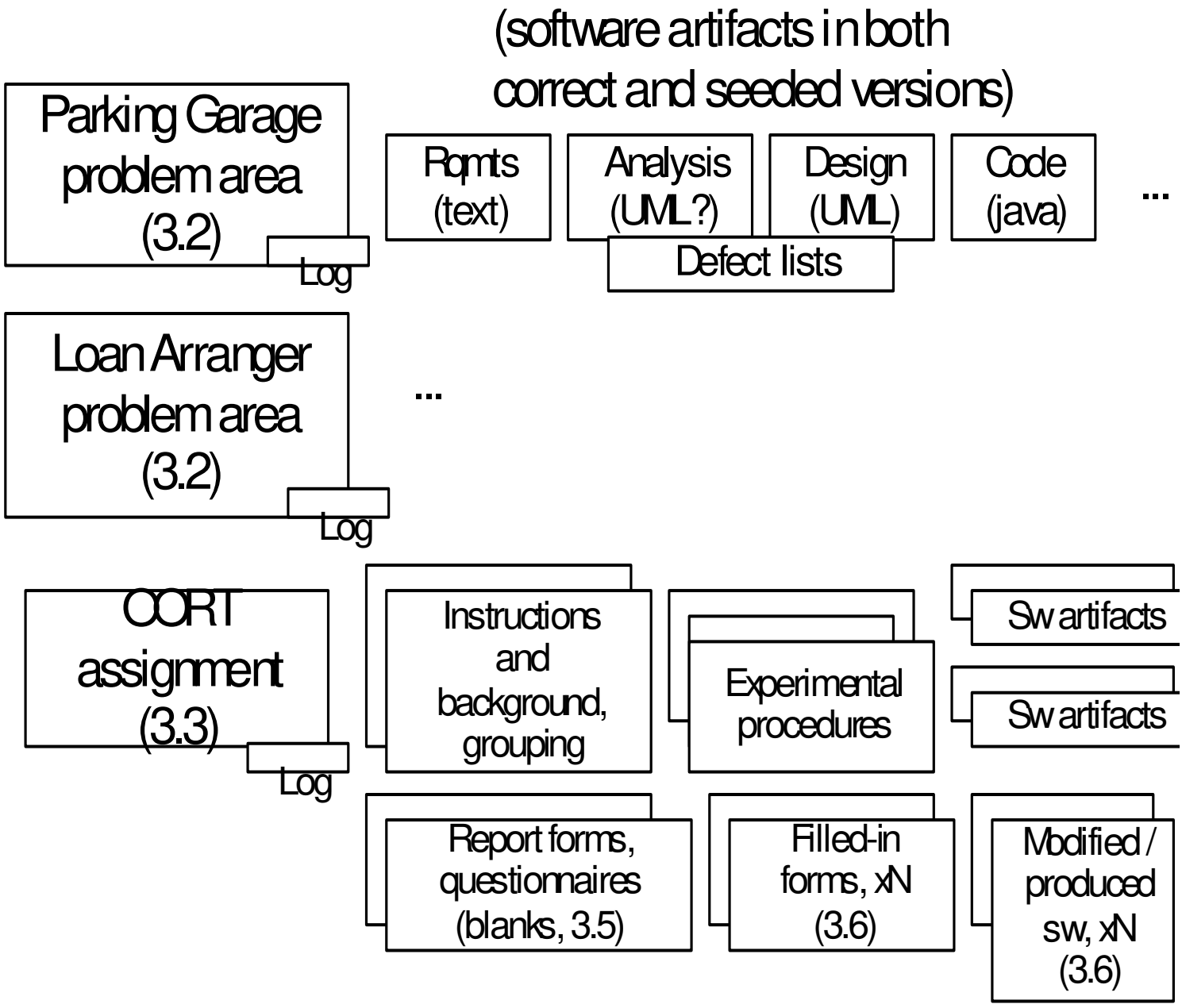
Some **common attributes** of a document are:

Type, choice of natural language, name (e.g. file name), responsible author and affiliation, title, abstract (optional), create date, last change date, version identification, possible links to other documents, contents (usually a text), and comment/change logs.

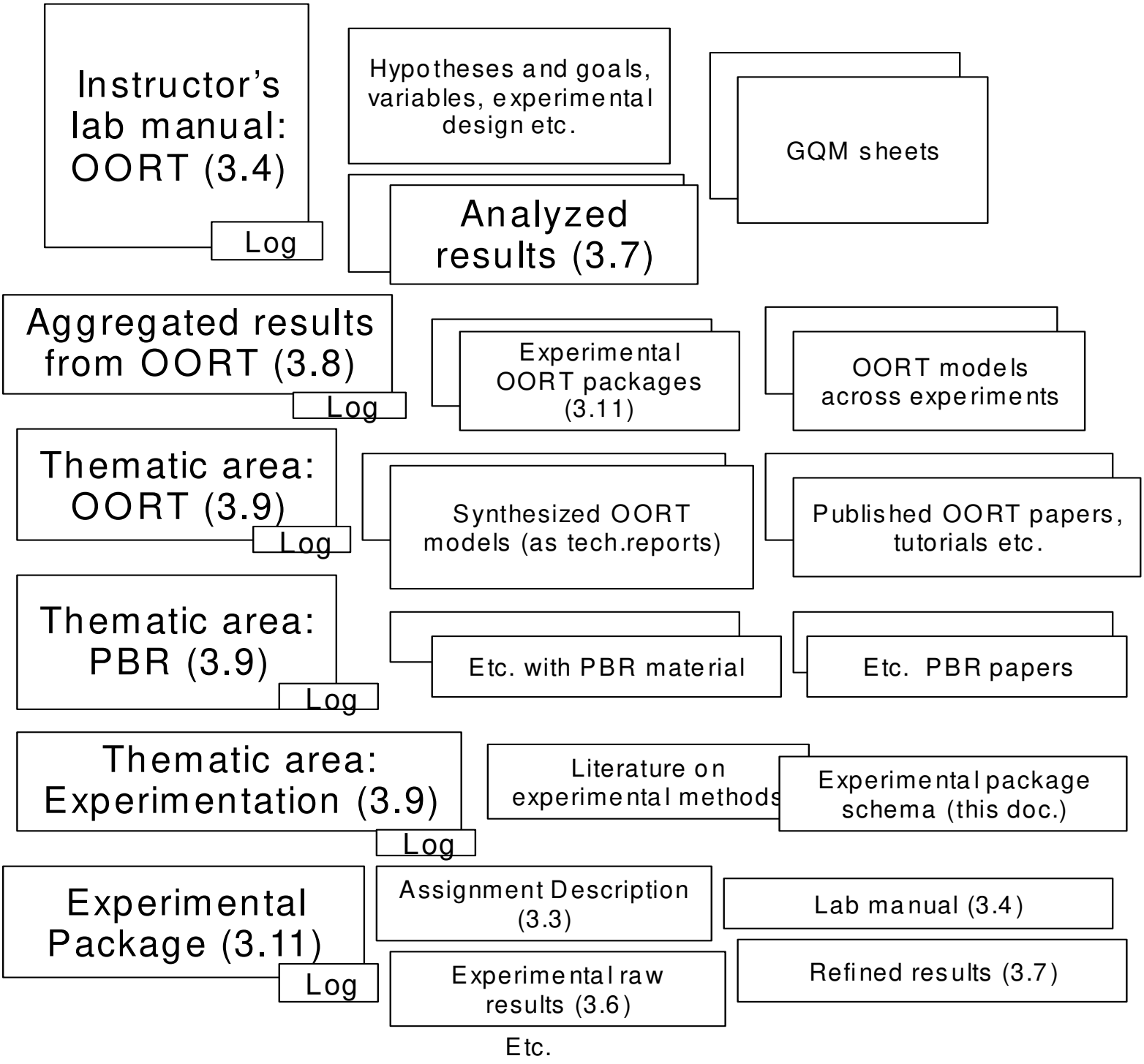
Other documents, e.g. Assignment Descriptions in 3.3 or Lab Manuals in 3.4, will have a recommended table of contents or items, as described for each document.

Parts of the documents will be described in a natural language different from English, but we expect that the **major documents and/or abstracts will be in English**, both to allow overall quality assurance and common access.

Some illustrations of typical document types are shown on the next two pages, with umbrella documents on the left and their components adjacent to the right:



GHT: also a graphical schema, or a formal document model?



3.2 Software Artifact

It is often *very hard* to get good, comprehensive examples -- cf. even the one at `www.rational.com` is not complete.
It is therefore beneficial with a shared pool of educational material, covering different problem areas (domains) and many usage situations. See IEEE homepage and some Canadian initiative??

Ex. There should exist a common ,but separate **Problem Domain** description (as a special unbrella document) for a set of related artifacts.

Ex. Some typical domains are Parking Garage (PG), Loan Arranger (LA), Gas Station Computer System (GSCS), Cruise Control (CC), Automatic Teller machine (ATM) etc.

Some artifacts may exist in different "sizes": small, medium, complete.
E.g. there may be a "small" and "complete" set of requirements for a GSCS.

Important that we explicitly specify the following attributes (as part of an abstract):
- goal or original purpose (see below), ambition level.
- rationale (why this domain was selected -- "I stole the PG from a textbook, because ...")
- origin: organization, person, when, why, ...
- contextual bindings -- hard! (cultural: American, domain: banking etc.)
- hidden assumptions (assumed level of domain knowledge needed), ...

Software artifacts have, and may be developed for, many actual uses, e.g.: -- see IEEE STD??
- education
- analysis (for defects)
- construction (maintenance, reuse) etc.

The concrete use context is also different:
- in courses (for illustration),
- in small individual exercises/assignments (for learning),
- in larger group projects (for learning),
- in student experiments, or
- in industrial experiments.

Software artifacts may be single and composite, and some artifact subtypes may be
- Problem domain description (an umbrella artifact)
- Requirements documents (text + ER diagrams + flow charts)
- Domain or business models (from domain engineering)
- Analysis documents (as UML use cases + comments)
- Design documents (e.g. UML diagrams + textual class descriptions)
- Program text: interfaces and bodies, source and compiled
- Test data and test cases
- System documentation
- User documentation
- Process models
- Etc.

Question: Do we need to model/represent relations between these, e.g. PartOf, DependOn (UsedWhere), RealizedBy, ... (via html in the umbrella document, or via an EMS?)
And we should be able to **navigate** between these.

Versioning: see section 4.1.
Question: What about product families (structurally similar sets of artifacts with planned variations)?

Ex. PG_Req.doc, PG_UseCase.mdl/.doc, PG_Design.mdl/.doc, PG_ClassDesc.doc
PG_ReqCorrect.doc vs. PG_ReqSeeded.doc

Log on comments and changes/versions, e.g. from uses in experiments.

3.3 Experimental instruction for subjects (assignment description)

An Experimental Designer (2.6) or Replicator (2.5) is responsible for making or updating an assignment description, as for a related lab manual (3.4). The main audience for an assignment description is subjects (2.8). It should contain the following information

* Context for experiment:
  project or course identification, exercise number, ...

* Goal(s):
  e.g. educational or industrial ones (at what detail level?)

* Input documents -- mostly electronic:
  - practical organization (from 3.4):
    group allocation: who is doing what (the so-called treatment) on which artifacts? etc.
  - selected set of software artifacts (3.2).
  - general background material (taken from 3.9).
  - special process instructions, i.e. **treatments** (operational guidelines from 3.10)
  - practical reminders and notes.
  - report forms (3.5): questionnaires (before/during/after experiment),
    defect/observation forms, comment and other data forms.

* Output documents, repeated for each subject/group -- all electronic? (stored in 3.6):
  - filled-in report forms
  - produced/modified software artifacts
  - other?

Ex. 735_a4-description.doc (for UMD, see Appendix D),
Exercise_OORT.doc (for NTNU, see Appendix E).

Log on comments and changes/versions, e.g. from uses in experiments.

3.4 Internal Instruction for researchers (lab manual or experimental plan)

To FC-MD team: especially read and comment on this (see also Appendix B)!!
See AAP Guidelines by Jeanice Singer (Appendix A), and also the set-up in the TR on OORT (CS-TR-4070, Oct. 1999).

An Experimental Designer (2.6) or Replicator (2.5) is responsible for making or updating such a lab manual. A lab manual must deal with and describe the following subjects:

Introduction: Why do this empirical study and how, in what context etc.:
- Theory: What has been done before, coupling to previous work etc. (literature references).
- Rationale, general goals etc.
- Concrete goals with experiment:
  * Research, pedagogic and/or industrial goals.
  * Hypothesis to be tested, with what variables etc. (see details later).
- Context for this experiment:
  * What are the subjects (see below)?
  * What are the artifacts and other equipment (see below)?
  * What are the procedures to be tested (see below)?
- Experimental design (see below).

Subjects: describe these, e.g. course participants or industrial developers, and comment upon their selection.

Software artifacts: list all artifacts to be used, and explain and justify their use.

Procedures/guidelines/"treatment" (process mechanics) of the experiment:
- what to be done before, during and after the experiment, and special reminders in all this.

Special requirements and assumptions (make explicit in order to motivate the experimental design below):
- Hypotheses/criteria to partition the subject groups, such as (in)homogeneity based on competence (via questionnaire) and possible pairing in executor/observer (e.g. for OORT) etc.
- Need for group stability or dependencies over several experiments?
- What artifacts to use for what groups (some may or may not require domain knowledge) etc.?
- Hints on what to tell and not tell students, e.g. how detailed should the treatment guidelines be?
  Ex. How much of goals and rationale should be revealed? (JC: this is more general!!)
  Ex. What should be said about schedule/effort guidelines?

Experimental design:
- Which subjects in what groups does what (treatment) on which artifacts?, i.e. an *allocation matrix* for student x treatment x artifacts.
- For class assignments only: Give overall plan for what lectures and exercises go together.

Independent and dependent variables, including context variables:
- subject experiences
- context of experiment (students, industry etc.),
- variability between subjects
- variability of events and actions

Hypothesis testing details -- questions and metrics:
- Filled-in GQM abstraction sheets to obtain operational metrics (these will define parts of the experiment instructions in 3.3).

Questions we cannot answer, but would like to:
- e.g. cannot have the ideal case, is impractical to test this, ...

Open questions: what areas we want feedback/data on, but cannot get? (but by replication we may be able to get more …).

Threats to validity: mention and discuss threats to external, internal, and concept validity.

Ex. 735_experiment.doc (by Forrest Shull et al., see **Appendix B**).

Log of comments and questions/answers, and subsequent changes.

3.5 Report and data form

These are forms (templates) to be filled in by subjects or researchers before/during/after the experiment. They include defect forms with documented defect categories and possibly severity, data sheets to record special measurements (e.g. for effectiveness), questionnaires and more qualitative comment forms. A general pattern for such templates must be given.

All forms are preferably in an electronic format, as a structured text, spreadsheet or similar. See also at the UMD web tool to collect data here.

Ex. 735_a4_Questionnaire.doc,
735_a4_Defect_1.doc, 735_a4_Observation_1.doc.

Log of comments and questions/answers, and subsequent changes.

3.6 Experimental raw results

Filled-in report forms (from 3.5 etc.), possibly stored in a database or as spreadsheets, and produced/modified software artifacts (may be taken from 3.2).

These data will be managed by the original Experimental Designer (2.6) or Replicator (2.5), and used by these and possibly by the Aggregator (2.7).

The data may be marked with confidentiality (persons, companies).

Must discuss how such material is available to people outside the core research team!

Log of qualitative experiences and comments made by researchers.

3.7 Experimental refined results (analysis)

These data will use data from 3.6, and providing data to be synthesized in 3.8.
They will be managed by the original Experimental Designer (2.6) or Replicator (2.5), and used by these and the Aggregator (2.7).

The data format is technical reports/papers, containing new/refined models.

Open issue: How open can/shall we be for initial results? May anyhow need to hide confidential stuff.

Ex. Initial notes/slides on data distributions (e.g by Jeff Feb. 28'2000), various comments, sketchy models etc.

Ex. OORT_TR_nnn/99.doc, paper-ICSE2000.doc
Ex. UMD-CS-TR-4070 for OORT results in 98-99.

Log of comments and initial hypotheses, and later discoveries (gradually aggregated into 3.8).

3.8 Aggregated results/models across families of replicated experiments (synthesis)

This will mainly be carried out by the Aggregator (2.7), in cooperation with the responsibles for the original experiments (2.6, 2.5).

Possible contents are:

The goals and hypotheses for combining results from a family of repeated or similar experiments.

Goals and hypotheses for the individual experiments (cf. 3.4).

As a result, it will contain synthesized material from individual experiments, discussing their variability and validities, concrete results and their possible interpretations and fallacies, possible conclusions, and generalized and/or revised models and insights.

Conclusions and recommendations should be drawn and fed back as a revised template for a family of future experiments, and fed back into 3.3 and 3.4 etc.

Log of insights/comments and how hypotheses/theories have evolved.

3.9 Thematic area: own and other material on some selected competencies

Thematic areas include general themes such as:
- experimental methods (GQM),
- data analysis techniques, such as statistical techniques,
- experience reuse (EF), etc.

They also include more specific technical areas such as:
- reading: PBR, OORT, …
- COTS,
- estimation, ...

Cf. also core competence areas and thematic blocks at FC-MD, lying as a scientific base across several (families of) experiments.

Such themes and associated material represent generalized experiences being gathered and processed from 3.6-3.8. The material is in the form of problem statements, state-of-the-art-summaries, technical reports, papers, slides, references and bibliographies to relevant literature etc. All this represent synthesized software models from experiments. The material can be on paper form and stored electronically via web, or combinations.
All this needs to be referred from assignment descriptions and lab manuals.

Special summaries and slides about this can be found in 3.10 below.

Log on comments of relevance and well-suitedness, and of changes.

3.10 Course/training notes and guidelines (for subjects)

This is tailored background material, with definitions of relevant concepts and terms and operational guidelines (based on material in 3.9), to be used as part of the experimental instructions in assignments (3.3).

Such material may constitute part of the support texts for a course, or material taken from a company SPI program. We may often use similar material for students and industry?

As handouts, slides etc. to supplement more "heavy" background material from 3.9.

Ex. summary_xx.doc, lecture_XX.ppt.

Log on comments of relevance and well-suitedness, and of changes.

3.11 Experimental package (a composite)

This includes a complete and separate set of the following material:

- General and experimental instructions for subjects (from 3.3, tailored for experiment): including blank report forms (3.5) and guidelines (3.10).

- Subset of software artifacts (given in 3.3, reused from 3.2).

- Internal instructions for researchers (lab manual from 3.4, tailored for experiment): including hypothesis, experimental design, metrics, threats to validity, …

- Experimental raw results -- preferable electronic (3.6): filled-in report forms and possibly produced/modified software.

- Experimental refined results, for analysis (3.7): reports, papers etc.

- References to more final aggregation papers (3.8) for synthesis, and finally to thematic areas (3.9).

How to couple related experiments (e.g. PBR and OORT) -- via aggregated experiments in 3.8?

We must also allow parts of this to be available to different colleagues, such as General Reader (2.4) vs. Replicator (2.5) vs. Aggregator (2.7).

Log on comments of relevance and well-suitedness of above structure, and of performed changes.

3.12 List of cooperating partners

This is a list of partners in academia and industry, with contact person, institution, postal address, phone, fax, email, and web address.

We should make a simple matrix over who is doing/responsible for what.

E.g. UMD, FC-UMD, NTNU, IESE, U.Oslo, U.Lund, Brazilian (COPPE/UFRJ, USP/Sao Paolo, UNIPACS), other ISERN partners etc. -- and with descriptions of their roles including responsibilities.

3.13 Glossary

This is intended as a glossary across classes of experiments (should later put in an Appendix).

General definitions:

Validity: define external, internal, and construct validities.
Types of empirical studies: experiment, case study, ...
...

Special glossaries:

Defect or discrepancy categories: omission, incorrect fact, extraneous, ambiguous, inconsistent.
Defect severities: e.g. minor, major, supermajor.

OO categories of behavior: functionality, service, method.

Role types:

2.1 Master Librarian
2.2 Steering Board
2.3 Theme Librarian
2.4 General reader
2.5 Replicator
2.6 Experimental designer
2.7 Aggregator
2.8 Subject

Document types:

3.2 Software Artifact
- Problem domain description (a "master" or umbrella artifact)
- Requirements documents (text + ER diagrams + flow charts)
- Domain or business models (from domain engineering)
- Analysis documents (as UML use cases + comments)
- Design documents (e.g. UML diagrams + textual class descriptions)
- Program text: interfaces and bodies, source and compiled
- Test data and test cases
- System documentation
- User documentation
- Process models
- Etc.

3.3 Experimental Instruction for subjects (assignment description)
3.4 Internal Instructions for researchers (lab manual)
3.5 Report and data form
3.6 Experimental raw results
3.7 Experimental refined results (analysis)
3.8 Aggregated experiments across a family of replicated experiments (synthesis)
3.9 Thematic area, covering families of studies (both general and technical)
3.10 Course/training notes and guidelines (for subjects)
3.11 Experimental package
3.12 List of cooperating partners
3.13 Glossary

Log on comments of relevance and well-suitedness, and of performed changes.

4. Organization of files/documents

4.1 General document organization

Each document will have a main **creator/responsible.** As mentioned in (3.1), related documents are organized in document **clusters**, having an **umbrella document** (in html?) as a "parent", and a shared **comment/change log** covering the whole cluster. The following umbrella documents is at least needed:

- **Problem domain** (3.2, with its software artifacts).
- **Assignment description** (3.3), with related artifacts (subsets of 3.2).
- **Experimental package** (3.11, with links to assignment description in 3.3, lab manual in 3.4, and experimental results in 3.6 and their analysis in 3.7). Each package must be a *self-*

contained entity, i.e. with *separate copies* of all files and data used -- so that we can securely preserve the experimental context for eternity.

- **Thematic area** (3.9), with coupling to aggregation in 3.8 and guidelines in 3.10.

We will have to establish a **federated, decentralized web cite**, where central documents are stored both locally and at cooperative partners (but avoiding double storage whenever possible). Such a web structure is both serving researchers locally and remotely, students and general web surfers (see 4.2 below on Portals).
Hint: try to reuse some existing web/file structures?

NB: must take care not spreading experimental set-ups to the community at large and thus invalidate future experiments.

Questions:
- Versioning: how to manage variants and revisions? -- hopefully by multiple copies of documents and with reasonable file names. We often make a (trivial) copy for a new experiment.
- How to assign document responsibles?
- What process should be used to manage a log of comments, frequently-asked-questions and related changes?
- Look at a future EMS in all this?

Hint: Look at BSCW? Sharing by symbolic or html links? What levels of file access granularity?

4.2 Portals

There will be many access ways or **portals** into this "experience base" or **Experience Library,** cf. a future EMS. The most relevant ones are:

- A **main portal to the Experience Library**, e.g. at IESE or at FC-UMD.
  This portal will contain the Documents Standard, ribbon web structures, web templates for documents, and the Quality Assurance Rules. It will list the Steering Board and the Master Librarian. It also will contain a survey of the cooperating cites and their participants, the currently accepted themes, and links to the local cite libraries.
  Readers: all.
  Writers: the Master Librarian.

- Via selected **problem areas** with related software artifacts (e.g. the NAWUS network).
  Readers: trusted colleagues (internal, external).
  Hint: reuse the SPC?? example at UMD.
  Writers: local teachers and experimenters.

Ex. http://fc-umd.edu/reading/reading.html.

- Via the **main web site** of the organization, leading to **main thematic areas** for general presentation and to synthesized results from families of repeated experiments (e.g. FC-MD web site).
  Readers: all interested, including other colleagues.
  Writers: theme responsibles at a local cite, partly Aggregators.

- Via (main web site and) **thematic areas**, here opening up for colleagues to **access more technical information** about past experiments (e.g. ISERN web page).
  Readers: cooperating Replicators.
  Writers: also Replicators (?).

- Via (main web site and) **course descriptions**, again leading to **assignment descriptions** (e.g. at most Computer Science Departments).
  Readers: local subjects
  Writers: local Experimental Designers or Replicators.

- Via **experimental packages**, but only with access to assignment descriptions and lab manuals (e.g. at FC-MD/UMD reading team).
  Readers: Replicators, local team members.
  Writers: Experimental Designers, Replicators (partly).

- Via **experimental packages**, but with full access to all raw and refined data (e.g. also at FC-MD/UMD reading team).
  Readers: Aggregator, trusted team members.
  Writers: Aggregator.

4.3 File naming conventions

File name conventions (not complete!):

- Problem area: uniquely abbreviated into a few capital letters, e.g. Parking Garage = *PG*.

- Source documents:
  PG_Req.doc, PG_UseCase.mdl, PG_Design.mdl, PG_ClassDesc.doc, …
  (see below on seeding)
  PG_ModuleXHeader.h, PG_ModuleXBody.c,
  PG_ModuleYHeader.java, PG_ModuleYBody.java.
  Also documents in *.ppt, .tex, ...*
  Seeded source variants: *PG_ReqSeeded.doc,* Correct ones: *PG_ReqCorrect.doc.*

- Derived documents (the usual case for reading): *PG_Reg.ps (or .html or .pdf), .obj, .dvi, etc.*

- For assignment descriptions: the suffixes "Seeded" or "Correct" must be removed!
  Mostly *.ps* or *.pdf* files, unless when filling in a report form where source files in the form of *.xls* or *.doc* files are needed.

- For change responsibles and dates: *OORT-labmanual-20mar2000-conradi.doc.*
  NB: Must consider names of global vs. local documents.??

Versioning: local copies per experimental package, possibly main revisions in the main repository (see 4.1).

Log of comments and questions/answers, and subsequent changes.

5. Intellectual (Legal) Rights

A general rule: owner/creator/home organization has the basic rights, implicit by copyrighting (by © in the US, by mere author name in Norwegian).

Try to get the effects of the "Free Software" community, so promote reuse on simple and generous terms. I.e. aim for general reuse of research/educational material in software engineering:

- Much material can be read as .html/.ps/.pdf (with copyright clauses).
- Some material can be read as .ps/.pdf, assuming comments are given back to the authors.

- Most material can be read in source form by "certified" colleagues for use in own courses/research, on the condition that all changes and research results are communicated back to the original authors. A contract on beforehand may be needed here.
  May need a common format for citations and acknowledgements.
- Some material is even more restricted here?

- Commercial use must always be agreed upon with the authors, e.g. courses for industry.

- Use of industrial results may be restricted, and require special agreements.

Examples of clusters of friendly communities: ISERN, NAWUS (see below), possible new ESERNET etc.

Ex. Norwegian NAWUS project (Univ.Oslo, NTNU + friends), see www.ifi.uio.no/~nawus:
- About ten Norwegian universities and regional colleges have variants of a software engineering or information system course. Some have emphasis on requirements/usability, others on design, others on the entire process.

- We have pragmatically classified the sw.eng. field in a dozen themes (cf. Sommerville's chapters): e.g. requirements, analysis, design, UML, SCM, QA, SPI, reuse etc. and with html links to the local educational material in the form of lecture notes and similar stuff.
- All partners can (re)use this in .ps/.pdf form (while observing copyrights). Thus all local material is viewed as a **common pool** of relevant **educational material.** Teachers can access source versions, but all source changes must be reported back to the authors and to the commonly accessible pool.
- Little exercise/project material yet, but often available by web sites for individual courses.
- Big need for complete, manageable and annotated examples!

Log of comments and questions/answers, and subsequent changes.

6. Examples

The below subsections will be filled later with more detail.

6.1 PBR experiments at UMD

See local UMD material on this, Forrest Shull's PhD thesis etc.

6.2 OORT experiments at UMD

See section 8: Appendix B on OORT lab manual at UMD and section 10: Appendix D on OORT Assignment Description at UMD -- both for course CS735 in fall 1999.
See also CS-TR-4070, Univ. Maryland, Oct. 1999, 36 p.

6.3 OORT experiment at NTNU

See section 9: Appendix C on OORT Lab Manual at NTNU and section 11: Appendix E on OORT Assignment Description at NTNU -- both for course 78038 in spring 2000.

6.4 Maintenance experiments at Univ. Oslo

See work by Dag Sjøberg and Magne Jørgensen (www.ifi..uio.no): one with 2nd year students modifying a 60,0000 lines C ++ program for numerical software, and one with 4th year students to modify a smaller program either in a OO/structured way or a procedural/non-structured way.

See also Magne Jørgensen's PhD thesis from Oct. 1993.

6.5 Reading/testing experiments at Univ. Kaiserslautern

See work by Christopher Lott and Oliver Laitenburger.

6.6 Reading/testing experiments at Univ. Lund

See work by Claes Wohlin et al.

6.7 Reading/testing experiments at Univ. Bari

See work by Guiseppe Visaggio and Filippo Lanubile on reading techniques.

6.8 Etc.

7. Appendix A: Jeanice Singer: APA Style Guidelines for reporting experiments

From Jeanice Singer, National Research Council, Canada: "Using the American Psychological Association (APA) Style Guidelines to Report Experimental Results", 5 p.
See http://wwwsel.iit.nrc.ca/~singer/

The paper outlines how experience papers on experimentation should be written. Most of thus should be relevant for the Experimental Designer (2.6) role? The items are:

1. Abstract
- Summary of the paper: hypothesis, method, studied population, and result overview.
- An abstract has two purposes:
  1) to allows a reader to decide if the article is interesting,
  2) to allow researchers to locate and retrieve articles (by keyword searchers).
- The abstract must be useful as a free-standing document.
- The results should be reported without evaluative comments!
- APA Style Guide: maximum 960 characters (i.e. less than half a page).

2. Introduction
- Describe the specific problem under study.
- Give a summary of what happened in the study and why, e.g. discussing questions regarding the point of the study, the relationship between the experimental hypothesis and design, and the theoretical implications of the study.
- Give a literature survey, to motivate the given study, and NB: treat controversial issues fairly. Statements of facts are generally more acceptable than length excursions into opinions! Start out broadly, and then be specific as the current research is introduced.
- End with a list of the variables manipulated and a formal statement of the hypotheses tested. The rationale of each hypothesis should be clearly defined. This can include expected results.

3. Method
- Describe how the study was conducted, i.e. a "cookbook" for research.
Needed to assess validity of the study, i.e. whether the study can justify the drawn conclusions.
Allows other researchers to perform replications and meta-analysis.
- See also subsections 3.1-3.3 below.

3.1 Subjects/participants
- Describe the human subjects in the experiment:
  * Who participated using demographic variables such as age, education, experience etc.?
  * How many participants were there?
  * How were they selected, e.g. from a course, being paid etc.?
- NB: Experimental design, such as allocation into groups, are not discussed (see Procedure).
- "Subject" may also be a piece of software or a software process. It is important for replication that necessary information is specified, e.g. what company participated, were the company given free software etc.? This may influence the motivation of the subjects, and thus the

results of the study.

3.2 Apparatus/Materials
- Describe the "apparatus" used in the experiment, to the relevant amount of detail to allow replication or meta-analysis (e.g. the execution speed of the computer).
- All software artifacts must be described (or referred to), possibly in appendices.
- NB: Use of the apparatus/materials is reported in Procedure below.

3.3 Procedure
- Describe the steps in planning and executing the research.
- Experimental designs (grouping x treatments x artifacts) must be explained, as well as information on randomization, counterbalancing, and other control features of the design.
- Participant instructions (assignment descriptions) must be outlined.
- NB: APA Style Guide does not outline an explicit design section, but this may be included, and must have information to allow meta-analysis: dependent variables, all levels of independent variables, and whether they are within- or between-subjects.

4. Results
- Summarize the collected data. and how data was managed.
- Explain all quantitative and qualitative analysis, and match with corresponding hypotheses.
- Report inferential statistics, with the value of the test, the probability level, the degrees of freedom, and the direction of the effect.
  Descriptive statistics (e.g. mean, median, and deviation) may be included.
- In APA Style Guide: basic assumptions, such as rejecting the null hypothesis, are not reviewed. However, questions about the appropriateness of a particular test (justification) must be discussed.
- NB: no interpretation of results here (in below Discussion), just reporting.

5. Discussion (interpretation)
- Evaluate results and their implications, especially wrt. the original hypothesis.
- Try to answer questions, e.g. how the original problem has been resolved and what conclusions can be drawn.
- Also briefly discuss the difficulties encountered then conducting the study, or unexpected results.
- If several experiments are presented, summarize both each experiment and the aggregate generalizations that can be drawn.

6. References
- This describes the APA reference style (RC: ignored here).

7. Appendices
- Misc. tables and figures -- but AP Style Guidelines states it is too early for giving any guidelines here.

8. Appendix B: OORT Lab Manual at UMD, course 735, fall 1999

Proposal for 735 Experiment (received by F. Shull et al., Feb 18, 2000, verbatim from file 735_experiment.doc)

Experimental Goal:
To get experience with, and help debug our methods for, running observational studies. (Especially the idea of Process Observer/Executor pairs.)
To get qualitative feedback on both PBR and the OO Reading Techniques (OORTs).
To understand the role of reader experience in applying OORTs.

Pedagogical Goal:
To introduce students to software experimentation.
To demonstrate to students the difficulties of thinking about software processes, and give them experience with a strategy that may be applied (observational techniques).
To familiarize students with the idea of evolving processes based on feedback.

Experimental design and treatments:
All estimates of numbers of teams in this document are based on an assumption of 24 people in the class.

	Treatment 1 (using PBR)	Treatment 2 (using OORTs)	Experience in treatment 2
Group A (3 teams)	Inspect Loan Arranger reqs (using T & U Techs)	Inspect Loan Arranger design (using Set 1 Techs)	Low domain knowl; Experience with reqs
Group B (3 teams)	Inspect Loan Arranger reqs (using T & U Techs)	Inspect PG design (using Set 2 Techs)	High domain knowl; No exp with reqs
Group C (3 teams)	Inspect PG reqs (using T & U Techs)	Inspect Loan Arranger design (using Set 1 Techs)	Low domain knowl; No exp with reqs
Group D (3 teams)	Inspect PG reqs (using T & U Techs)	Inspect PG design (using Set 2 Techs)	High domain knowl; Experience with reqs

We assume that students assigned to read the Loan Arranger requirements have *little* domain knowledge in that domain, and that students assigned to read the Parking Garage (PG) requirements are *very comfortable* with that domain.

Possibly confounding variables to be controlled for:
- Previous domain knowledge
- Knowledge of notation (English in treatment 1; the OO paradigm in treatment 2)
- Team composition (is the process observer or process executor the more experienced?)

Each of these variables has to be balanced as much as possible (i.e. we can't really randomize).

Mechanics of the Experiment:
- Pre-test questionnaires must be filled out by the students to assess their experience with:
  - the "Loan Arranger" domain (steal questions from Marcus' experiment);
  - using PGs;
  - the OO paradigm and terminology (steal questions from previous 435 experiment);
  - English;
  - The specific PBR techniques that are to be used (e.g. use cases, testing; steal questions from previous 735 experiment).
- Consent forms will need to be distributed.
- Students will be assigned to work in two-person teams. Assignment will be done quasi-randomly (using the constraint that at least one person on each team have some prior exposure to OO).

- One student will be given the job of "process guide and observer" while the other will be the "process executor." The process observer will be required to step the executor through the steps of the process and collect data on the executor's application of the technique.
- The process observers will be given a copy of the procedure the executor should follow, with one step per page. Each step of the procedure will be followed by a number of observational questions, which the process observer must fill out as the executor works, and a number of post-hoc questions, which the guide must ask the executor as soon as the step is completed. The role of process observer is mostly passive; the observer should interfere as little as possible with how the process executor performs, but should only remind the executor what step he or she should be on.
- The students will switch roles between treatments.
- Because the process observer is meant to be a largely passive role, if only one member of the team has sufficient experience in a particular area (e.g. experience using OO in treatment 2) that member should be the process executor.
- Each treatment will result in:
  - Qualitative data concerning how the technique was applied.
    - Synthesis or abstraction of raw data by Process Observer
    - Raw data (allows us to have some backup in case synthesis/abstraction of poor quality)
  - A list of defects resulting from application of the technique.

- Treatment 1:
  - 2 of the PBR techniques will be applied (probably user and tester, since "designer" perspective might interfere with design review in treatment 2). One technique will be applied by each team.
  - 2 different requirements documents will be used: one in a familiar application domain (PG), one in an unfamiliar domain (Loan Arranger). This will allow us to see if different issues are reported by the observers depending on the amount of domain knowledge of the executor (see "Questions and Metrics").
- A questionnaire given to subjects between the treatments should be designed to allow us to correct mistakes in the experimental method before applying it again.
- Treatment 2:
  - 2 sets of OORTs will be applied. One set of techniques will be applied by each team, with the set of techniques used corresponding to the document being inspected. Set 1 will be used to read the Loan Arranger, since the state diagrams are not as important for this document, while Set 2 will be used to read the PG. Although this results in a confounding effect (set of techniques confounded with document read) it increases the number of data points that can be compared. The sets are composed as follows:
    - Set 1 (4 techniques): Req x Class Desc; Class Desc x Class Diag; Class Diag x Interaction Diag; Interaction Diag x Use Cases
    - Set 2 (4 techniques): Class Desc x State Diag; State Diag x Interaction Diag; Interaction Diag x Use Cases; Req & Use Cases x State Diag
  - 2 different designs will be used: one from the PG problem, and the other from the Loan Arranger. This will allow us to see if different issues are reported by the observers depending on whether the executors have previous domain knowledge (i.e. whether they reviewed the requirements from the corresponding domain in treatment 1).
- Follow-up:
  - Questionnaire for feedback on global process issues
  - Change the PBR techniques based on class feedback and redistribute (?)
  - Class discussion (?)

Questions and Metrics:

- Does having the process observer/guide affect the results of the process?
- Can measure what percentage of (known) defects are reported by teams reviewing the PG requirements (6 teams), and compare this result to historical data for these reviews.
- Do readers who have experience with the requirements find this an aid to finding defects in design?

- Compare qualitative (look for different types of problems) and quantitative (see if they find different numbers/types of defects) data for teams who read requirements and design from the same problem domain (6 teams) to the data for those who read documents from different domains (6 teams).
- Compare the *types* of defects found by each set of teams (6 teams/set) to determine whether having experience with the requirements results in a different number of defects related to the problem domain. (This requires use of a defect taxonomy that labels defects depending on whether they result from the problem space or solution space.)
- Does PBR work more or less effectively when the reviewer has some experience with the problem *domain*? (PG vs. loans)
  Does PBR work more or less effectively with the different *styles* of requirements (PG vs. LA)?
- We can't test these questions explicitly, but we can see if characteristic problems are reported from the observational data for each team (12 teams).
- What can we do to improve PBR?
  What can we do to improve the OORTs?
- We will ask the process observers a set of questions to fill out during the observational study. (This will give us 6 teams' feedback for each PBR and OORT technique used.) For example: Does the reader think a particular step is worthwhile? Can he/she suggest other techniques or methods that are commonly used to achieve the same goal as this step?
- Is the distinction between horizontal and vertical techniques in the OORTs a valid one?
- Measure whether the types of defects found by each team (12 teams) are correlated with the type of reading technique (horizontal or vertical) used.

Questions We Can't Answer, But Would Like To

- What *types* of defects occur in OO designs?
- We can't answer this because, even in the most ideal case, we would only have one design for each system. From this we could only reason about what types of defects one designer (or design team) makes, not about general classes!
- Trying to answer this type of question would impose impractical constraints on the experiment, e.g. we wouldn't be able to use the defect taxonomy to teach students what to look for without biasing the experiment!

Requirements:

- We expect only one-half the class to have been exposed to OO previously.
- Maybe we can check this by sending email questionnaires to students already registered for the class????
- The treatments will be done as two homework assignments of approximately 4-5 hours each. Grading:
- Quality and specificity of defect descriptions (not number of defects)
- Quality of synthesis of observational data
- We will require some class time to perform training in the reading techniques and in the observational method.
- Training Process Observers should occur in class. Probably 2 instructors should demonstrate the process to the class; the notes should be shown to the class afterwards. The demonstration should be on some other process (i.e. we don't want to bias them by demonstrating on a reading technique.) We should also aim for a one-page guide or reminder of the process.
- 1 class will be set aside for training for each treatment. Students should be given an example to work through. (This training will also have to include how the process observers should fulfill their responsibilities.)
- Training will emphasize defect *detection*, not *correction*. We have to emphasize that defects must be reported as specifically as possible, so we can use these for abstracting a defect taxonomy.
- An additional class may be necessary for training in OO.
- The PG and Loan Arranger requirements (treatment 1) can be used as is. We already have a defect list for each.
- For the Loan Arranger we can use Guilherme's design, which will require only minor changes (relatively speaking). We will use Rational Rose to try to take out any obvious inconsistencies.

- The PG design (treatment 2) also requires work. Jeff will design the PG from scratch (independent of the Rumbaugh book) since presumably there will be less danger of him unconsciously using the reading techniques to bias the way he designs.
- ESEG/FC-MD members will take part in the experiment. We can't really use their list of defects (since presumably they have a different understanding from the rest of the class of "defects" at this point) but we can use observational data. There is no reason we can't use for them the grading scheme outlined above.

Open Questions:

- What specific areas of PBR and the OORTs do we want feedback on? As a result, what questions should we ask the process observers to answer?
- ~~Does Jeff take part in the experiment? Maybe yes – we can't use his list of defects but can probably use his observational data. How do we grade him? Do we assign him to the Loan Arranger requirements (but he already reviewed these in 435) or to the PG requirements (but he already has Loan Arranger domain knowledge, so he might bias the design review treatment)?~~Presumably Jeff will review Loan Arranger requirements again (since he fixed the PG requirements, is doing the PG design now, and already did a LA design last year)?

Class Assignments

1. Literature review
2. PBR observation & synthesis
3. OORT observation & synthesis
4. Redesign the experiment, based on their experiences, threats to validity, …

Addendum 1 (RC): course 735 is a MSc/PhD level course at UMD, with 40?? students.

Addendum 2 (RC): Experimental artifacts/documents -- from web page of course:
Inputs:

- *Lecture slides (2-3 relevant sets)*
- *Assignment description (Appendix D for UMD) -- with defect classification table*
- *Questionaire.doc (filled-in before the reading)*
- *Group allocation (based on questionnaires)*
- *Definitions of terms*
- *Defect_1.doc, Defect_2.doc, Defect_3.doc, Defect_4.doc, Defect_5.doc, Defect_6.doc, Defect_7.doc*
- *Reading_1.doc, Reading_2.doc, Reading_3.doc, Reading_4.doc, Reading_5.doc, Reading_6.doc, Reading_7.doc*
- *Defect_obs_1.doc, Defect_obs_2.doc, Defect_obs_3.doc, Defect_obs_4.doc, Defect_obs_5.doc, Defect_obs_6.doc, Defect_obs_7.doc*
- *LA_Req.ps, LA_UseCase.ps, LA_Design.ps, LA_ClassDescr.ps*
- *PG_Req.ps, PG_UseCase.ps, PG_Design.ps, PG_ClassDescr.ps*

Outputs:

- *Filled-in questionnaires from all students*
- *Final reports from all student groups, containing:*
  - *Comments on own learning and possible difficulties*
  - *Comments on usefulness of OORTs*
  - *Suggestions for improvements/limitations*
  - *Defect reports*
  - *Observation reports*
  - *Effort (time usage) spent on applying each OORT*

9. Appendix C: OORT Lab Manual at NTNU, course 78038, spring 2000

See the lab manual for the UMD course (section 8: Appendix B), and the OORT technical report from Oct. 1999 (CS-TR-4070, UMD).

General goal: try to mostly repeat the course-735 experiment at UMD from fall 2000 at NTNU in a SPI/QA course for 4th year students in spring 2000, with ca. 20 students.

Research goals:
- Try out the seven, new OORTs (feasibility)
- Compare with UMD results from fall 1999
- Learn about and possibly improve experimental methods and the actual OORT experiment (for Reidar, at least!)

Educational goals:
- Let students learn relevant software engineering techniques

Metrics (incomplete):
- Time usage for actual reading
- Number of defects found and their types
- Qualitative comments on learning, feasibility and improvements

Differences, planned:
- Slightly operationalized the assignment description, to have the process is a bit more structured (goal: local tuning, simplification)
- Operationalized and corrected several minor errors (see Appendix G) in the seven OORT descriptions (goal: promote learning)

Hypothesis of differences to be investigated:
- Impact of cultural differences: Norwegian vs. American?
- Impact of lack of domain knowledge: e.g. PG may not be a known problem domain?
- Different knowledge level in software engineering/OO etc.?

Recognized problems:
- Two part-time teachers being responsible (some lack of commitment), a bit outside of scope of the TA's PhD topic (experience bases)
- Students may not devote enough effort and dedication, since this is a pass/no-pass assignment
- Insufficient time for local understanding, planning and coaching

Experimental artifacts/documents:

Inputs:

- Lecture slides: 2 sets (modified from 735 course)
- Assignment description (Appendix E for NTNU) -- without defect classification table
- Questionaire.doc (filled-in before the reading, as before)
- Group allocation (based on questionnaires)
- (Definitions of terms: stands in lecture slides)
- Defect_form.doc (common for all OORT1-7, including defect classification table)
- Observation_form.doc (common for all OORT1-7, taken from lecture foils)
- LA_UseCase.ps, LA_Design.ps, LA_ClassDescr.ps (all as before)
- PG_Req.ps, PG_UseCase.ps, PG_Design.ps, PG_ClassDescr.ps (all as before)

Outputs:

Filled-in questionnaires from all students
Final reports from all student groups, containing:
- defect reports
- observation reports
- comments on own preparations to understand the assignment (i.e. the OORTs)
- comments on usability of OORTs to find defects
- suggestions for improvements/limitation of OORTs
- time usage for readings

10. Appendix D: OORT Assignment Description at UMD, course 735, fall 1999

Received from Forrest ca. Feb 5, 2000: assignment 4(1).doc (.pdf on web page)

CMSC 735

Assignment 4

This assignment is a continuation of assignment 2, dealing with software process assessment. As before, your team will be given a procedure to be assessed, and must submit a **report** evaluating this procedure.

The Procedure to be Evaluated
You will be evaluating an Object-Oriented Reading Technique (OORT). An OORT is a procedure for inspections of OO designs. Your team will be assigned a particular design to which to apply the inspection procedure, and a particular set of OORTs that have been selected for that design. An overview of the OORT procedures will be presented in class on Nov. 1.

We will use the same taxonomy of defect types as in assignment 2. They can be tailored to OO designs as follows:

Defect	***General Description***	***Applied to design***
Omission	Necessary information about the system has been omitted from the software artifact.	One or more design diagrams that should contain some concept from the general requirements or from the requirements document do not contain a representation for that concept.
Incorrect Fact	Some information in the software artifact contradicts information in the requirements document or the general domain knowledge.	A design diagram contains a misrepresentation of a concept described in the general requirements or requirements document.
Inconsistency	Information within one part of the software artifact is inconsistent with other information in the software artifact.	A representation of a concept in one design diagram disagrees with a representation of the same concept in either the same or another design diagram.
Ambiguity	Information within the software artifact is ambiguous, i.e. any of a number of interpretations may be derived that should not be the prerogative of the developer doing the implementation.	A representation of a concept in the design is unclear, and could cause a user of the document (developer, low-level designer, etc.) to misinterpret or misunderstand the meaning of the concept.
Extraneous Information	Information is provided that is not needed or used.	The design includes information that, while perhaps true, does not apply to this domain and should not be included in the design

Evaluating the Procedure
In order to evaluate the procedure, you and your teammate will each be assigned distinct roles, Process Executor and Process Observer, with the same responsibilities as before. However, the roles will be switched from assignment 2, as specified on the accompanying table.

As before, both team members should write the final report, which is due in class on **Nov. 10**.

You Should Turn In:

1) A final report evaluating the usefulness of your assigned OORTs for achieving the task of defect detection.
   a) The report should be 3-5 pages in length, double-spaced.
   b) Your report MUST address the following topics:
      i) An in-depth explanation of the methods you used to understand the procedures, and your evaluation criteria.
      ii) Your assessment as to whether or not the OORTs were useful for the task they claim to address. If your answer is yes, what are the limits of your evaluation, that is, how broadly can you extrapolate your results? If your answer is no, are there any hints in your analysis of situations in which OORTs would be more applicable?
      iii) Does your analysis reveal anything about ways to improve the OORTs, either to make them *work* or to make them *work better?* Why or why not?
      iv) You may choose to compare the OORT procedure to the PBR procedure that you applied in assignment 2, to help clarify what you found beneficial or not about the OORTs.
2) The list of defects found by the Executor. A form for reporting defects will be placed on the class web page.
3) The notes taken by the Observer.

The due date for all items is Nov. 10.

Your grades will be based on: the quality of your final report, as determined by the instructor, and how well you conformed to the procedures that you were asked to apply (OORT and the Observer roles). Your grades will NOT depend on your specific answers, e.g. the number of faults that you report, or whether or not you found the techniques valuable.

NOTE: This assignment is part of a study. As always, working with another student will be considered cheating, but for the purposes of the study it is especially crucial that you do not discuss your work with other students in the class. The motivation and design of the study will be discussed in class later this semester.

11. Appendix E: OORT Assignment Description at NTNU, course 78038, spring 2000

Fag 45038 Programvarekvalitet og prosessforbedring
IDI, NTNU, vår 2000
Øving 2: Inspeksjon -- general

Object-Oriented Reading Techniques for design documents, deadline Friday March 3, 2000

Guilherme H. Travassos, Forrest Shull, Jeff Carver, Victor R. Basili, U. Maryland
Reidar Conradi, NTNU, Norway, p.t. Univ. Maryland

.

Given OORT task:

Goal: To learn about relevant OO Reading Techniques (OORTs) and to apply them to given software artifacts to find defects in OO designs (UML documents), to evaluate such techniques, and to compare the results with similar studies at Univ. Maryland.

Organization: Groups consisting of two persons (one reader and one observer) will be assigned a particular design to be reviewed, and a particular set of OORTs to be used for that design (either OORT-2,3,6,7 or OORT-1,4,5,6 -- see lecture foils of 21.2 and 22.2). The group composition and what designs and OORTs to be used by which groups stand elsewhere.

Steps:
1. Learn about the techniques (8 h, partly at two double lectures, partly self study).
   Fill out questionnaire, and be divided in groups.
2. Prepare for reading: e.g. read the software artifacts and especially the requirement document, re-read the selected OORT techniques etc. (2 h, mainly self study).
3. Read and observe (4 h, ca. 1 h per OORT, in groups). See guidelines elsewhere.
   Do the reading/observation in two rounds, as your concentration goes down after two hours.
4. Evaluate, sum-up and write final report (2h, in groups).

NB: The time estimates are just indicative; take the time needed to do a proper job!

Learning and evaluation emphasis: The emphasis is not only to find as many defects as possible, but also to *learn* and *evaluate* the given and novel OORT procedures. That is, both you (as students) will learn about software reading in general and OORTs in particular, and we (as teachers) will learn how to improve the OORTs. In order to evaluate the OORTs, your group is paired -- one Process Executor and one Process Observer.

Given input documents:

1. **Description** of Exercise 2 − i.e. this document (Exercise_OORT.pdf/.doc).
2. **Questionnaire** (Questionnaire.doc) to help to form groups.
   See separate information about **group composition** and given software artifacts and OORTs (on a separate sheet) per group.
3. **Theory material** for OORT and Observation, see the above foil sets (fag45038-OORT-21feb2000.pdf, fag45038-obs_training-22feb2000.pdf).
4. Given set of **software artifacts**:
   - Requirement documents: Requirement descriptions and Use cases.
   - Design documents: class, state and sequence diagrams, and class descriptions.
     There are two prepared problems: Loan Arranger and Parking Garage.
     Files: LA_Req.ps, LA_UseCases.ps, LA_CodeDesc.ps, LA_Design.ps and
     PG_Req.ps, PG_UseCases.ps, PG_CodeDesc.ps, PG_Design.ps.
5. **Templates** for **Defect list** (Defect_forms.doc).
6. **Templates** for **Observation notes** (Observation_forms.doc).

Expected output documents:

1) A **final report** evaluating the usefulness of your assigned OORTs for achieving the task of defect detection in design documents, 2-3 pages. Mark the report with your names and indicate who were Executor and Observer. Your report *must* address the following topics:
   i) An **explanation** of the **methods** you used to **understand** the OORT-procedures, and your own evaluation criteria for the OORTs.
   That is, what was unclear and how did you manage to clarify? How did you cooperate to prepare yourself for the reading/observation.
   What success criteria did you have?
   ii) Do you think the **OORTs were useful** for their claimed tasks?
   If your answer is *yes*, what are the limits of your evaluation, that is, how broadly can you extrapolate your results? If your answer is *no*, can you think of situations in which OORTs would be more applicable?
   iii) Does your analysis contain hints to **improve the OORTs**, e.g. to make them *work* or *work better?* Why or why not?
2) The **Defect list** made by the Executor − edit the given templates here.
3) The **Observation notes** taken by the Observer − edit the given templates here.

Acceptance criteria for this exercise:

Your report must be accepted by the course responsibles ("Bestått") as a precondition to take the final exam in this course. It is important that the Executor and Observer conform to the given procedures/roles (OORT and Observation). However, your report will *not* be evaluated based on your specific answers, e.g. the number of faults that you report, or whether or not you found the techniques valuable.

Note 1: Working with another group will be considered cheating. Since one major purpose is to evaluate the novel OORTs, it is especially important that you do not discuss your work with other groups in the class.

Note 2: This exercise is part of a study done jointly with University of Maryland in College Park and the Fraunhofer Center -- Maryland.

Comment log (March 7, 2000):

-- JC: Too much material for students (RC: but mainly adapting and merging existing material one).

12. Appendix F: The Original and the Refined OORT-4 -- Class Diagram x Class Description

See section 13: Appendix G for more comments about the changes being made..

@@@@@@@@@@@@@@@Old OORT-4 reading description:

Reading 4 -- Class diagrams x Class descriptions

Goal: To verify that the detailed descriptions of classes contain all the information necessary according to the class diagram, and that the description of classes make semantic sense.

Inputs to Process:

1. A class diagram (possibly divided into packages) that describes the classes of a system and how they are associated.
2. A set of class descriptions that lists the classes of a system along with their attributes and behaviors.

1) Read the class diagram to understand the necessary properties of the classes in the system.

INPUTS: Class diagram;
Class description.

OUTPUTS: Discrepancy reports.

For **each class on the class diagram,** perform the following steps:

- ☞ Find the relevant class description. Mark the class description with a blue symbol (*) when found.
  If you can't find the description, you have found a defect of omission. This class was represented but not described on the class diagram. Fill in a discrepancy report for this.
- ☞ Check the name and textual description of the class to ensure that they provide a *meaningful* description of the class that you are considering at this time. Also check that the description is using an adequate abstraction level.

  Can you understand the purpose of this class from the high-level description? If not, the description may be too ambiguous to be used for the design model. Fill out a discrepancy report describing the situation.
- ☞ Verify that all the attributes are described along with basic types.

  Are the same set of attributes present in both the class description and the class diagram? If not, it means that attributes are not appropriately described. This represents an inconsistency between the documents. Fill in a discrepancy report describing this situation and showing which document didn't capture the appropriate information.

  Can this class *meaningfully* encapsulate all these attributes? That is, does it make sense to have these attributes in the class description? Are the basic types assigned to the attributes *feasible* according to the description of the attribute? If not, it represents an ambiguity or

an incorrect fact. Fill in a discrepancy report describing this situation and showing which document didn't capture the appropriate information.

- ☞ Verify that all the behaviors and constraints are described.

  Are the same set of behaviors and constraints present in both the class description and the class diagram? Does the class description use the same style or level of granularity (e.g. pseudocode) to describe all of the behaviors? If not, you have discovered an inconsistency. Different information is contained in different documents, or different styles are used within the same document. Fill in a discrepancy report describing this situation and showing which document didn't capture the appropriate information.

  Can this class *meaningfully* encapsulate all these behaviors? Do the constraints make sense for this class? If not, it represents an ambiguity or an incorrect fact. Fill in a discrepancy report describing this situation.

  Do the behaviors accomplish this procedure using attributes that have been defined (for this or some other class)? Are the constraints satisfiable using the attributes and behaviors that have been defined? If not, you may have discovered an omission or ambiguity. The behaviors and constraints as defined cannot be satisfied using the attributes and behaviors that have been defined. It may be that new attributes must be included in the design, or the definition of the constraint/behavior changed. Fill in a discrepancy report describing the problem.

  Do the behaviors for this class rely *excessively* on the attributes of other classes to accomplish their functionality? (Note that you must make a value judgement about what is meant by "excessive reliance." You should compare the number of references to other classes for this class with the rest of the system, and consider the type of functionality addressed to determine if such reliance is really necessary.) If so, you have uncovered a potential style issue: unnecessarily high coupling. Fill in a discrepancy report describing the problem as a "miscellaneous" defect.

- ☞ If the class diagram specifies any inheritance mechanisms for this class**,** verify that they are correctly described.
  Is the inheritance relationship included on the class description? If not, you have uncovered a defect of omitted information. The class diagram specifies that this class is part of an inheritance hierarchy that should be described in the class description. Fill in a discrepancy report describing this situation.
  Use the class hierarchy to find the parents of this class. Is it true that, *semantically,* a <class name> is a type of <parent name>? Does it make sense to have this class at this point of the hierarchy? If not, you have uncovered a potential style issue: the hierarchy should not be defined in this way. Fill in a discrepancy report describing the problem as a "miscellaneous" defect.

- ☞ Verify that all the class relationships (association, aggregation and composition) are correctly described with respect to multiplicity indications.

  Were the object roles captured on the class description? Is the correct graphical notation used on the class diagram to denote the type of relationship? If not, you have discovered an inconsistency; the information on the two diagrams does not agree. Fill in a discrepancy report describing this situation and showing which document didn't capture the appropriate information.
  ***Semantically*, do the relationships make sense given the role and the objects related? For**

example, if a composition relationship is involved, do the connected objects really seem like a "whole-part" relationship? If so, you have uncovered a potential style issue: the relationships should not be defined in this way. Fill in a discrepancy report describing the problem as a "miscellaneous" defect.
If cardinalities are important, were they described in the class description? Given your understanding of the relationship, do the quantities of objects used seem enough? If not, you may have discovered an inconsistency. Fill in a discrepancy report describing the problem.
Is there some attribute representing the relationship? Does it use a feasible basic type, or structure of basic types (if multiple cardinality is involved)? If not, you may have discovered an inconsistency (if the documents do not agree). Fill in a discrepancy report describing the problem.

2) Review the class descriptions for extraneous information.

INPUTS: Class description.

OUTPUTS: Discrepancy reports.

☞ Review the class descriptions to make sure that all classes described actually appear in the class diagram.
Is there an unstarred class description? If so, you have discovered a defect of extraneous information. Class information has been described that does not actually participate in the class diagram. Fill in a discrepancy report describing the problem.

@@@

Step R4.1:Read the class diagram to understand the necessary properties.

- *Inputs:*
  - 1. Given class from class diagram (CD).
  - 2. A set of class descriptions (CDe).

- *Outputs:*
  - 1. Discrepancy reports.

- *Instructions:*
  - Q41.a: Is there a CDe for this class? [*omission?*]
    Mark with a star ("*") in **blue** on the CDe when found, see R4.2.

  - Q41.b: Is the name and textual description of this class meaningful in the CDe? [*ambiguity?*]

  - Q41.c: Are **attributes** and their types consistent between the CD and CdE?
    [*inconsistency?*]

  - Q41.d: Can this class meaningfully contain all these attributes and with given types? [*ambiguity?* or *incorrect fact?*]

  - Q41.e: On **behavior and constraints**:
    E.g. Check consistency for behavior and constraints between the CD and CDe.
    E.g. Are behaviors in the CDe described at the same level of detail / pseudocode? [*inconsistency?*]
    E.g. In general, should this class really contain all these behaviors and constraints? [*incorrect fact?*]
    E.g. Do the behaviors and constraints in the CDe use available behaviors or attributes from elsewhere, and are they defined? [*omission?* or a*mbiguity?*]
    E.g. Do the behaviors and constraints in the CDe rely "excessively" on attributes in remote classes? I.e. too high coupling? [*miscellaneous?*]

  - Q41.f: In case of use of **inheritance** in the CD:
    E.g. Is inheritance also included in the CDe? [*omission?*]
    E.g. In general, is it "meaningful" for the given class be a supertype/subtype of the iven subclasses/superclass? [*miscellaneous?*]

  - Q41.g: Check that all **relationships** are correctly described:
    E.g. Do they have the right cardinalities, and are they also defined in the CDe? [*inconsistency?*]
    E.g. We object roles in the CD also defined in the CDe? [*inconsistency?*]

E.g. Is the correct graphical notation used in the CD? [*inconsistency?*]
E.g. In general, do the stated relationships "make sense", such as composition vs. aggregation vs. association vs. inheritance etc.? [*miscellaneous?*]
E.g. Is an attribute used to represent a relationship, and does this have the right type (a reference or sets of references)? [*inconsistency?*]

Step R4.2: Review the class for extraneous information.

- ***Inputs:***
  - 1. A set of class descriptions (CDe).
- ***Outputs:***
  - 1. Discrepancy reports.
- *Instructions:*
  - Q42.a: Are there any unstarred (i.e. superfluous) classes in the CDe? [*extraneous?*]

13. Appendix G: Comments from Designing an OORT Experiment at NTNU

Initial comments on miscellaneous experimental OORT material from course CS-735 in autumn 1999 at FC-UMD/UMD, as inputs to a possible "experience base".

This is an expanded version over the email of March 2, 2000.
See also my email of 20 Oct 1999 on many of the below aspects.

A. OORT lab manual for UMD 735 course, autumn 1999

I had a problem in gathering and understanding all assumptions:

- E.g. Choosing the PG as the well-known domain may not be valid for European students.
- E.g. That students groups should not get too detailed process instructions on some points (does not stand in the lab manual). However, on some areas they get a lot of detailed guidelines, on some others not. Of course, as a general rule, they should know only what they need for the stated purpose, cf. also perpetual discussion on human subjects as "guinea pigs".
- E.g. Possible couplings to earlier PBR experiment are too implicit, as these are considered part of *one* experiment.
- E.g. Missed explicit rules on how students are grouped, based on questionnaire on OO/UML/industrial experience.
- E.g. Assumptions on how detailed the OORT instructions should be wrt. time estimates etc.

And, I got hold of this Feb. 18th, *after* the NTNU version was completed.
So, maybe all UMD lab manuals could be assembled and made available?

B. Available Requirement Descriptions: many and in different versions

Examples:

- *Parking Garage (PG):* 18 p. long, and hardly known to most Europeans.
- *Loan Arranger (LA):* unknown to European students.
- *Gas Station Computer System (GSCS):* I have seen a mini and a medium version, but even the latter does not explain all the sequence diagrams in the lecture slides.
- *Automatic Teller Machine (ATM):* Have not seen this.
- *Cruise Control (CC):* Have not seen this either, but seems very "American".

In general: these specifications tend too be very wordy.

C. Terminology, especially on Object-oriented technology

I have a problem with the behavioral levels:
- *functionality* ("end-user" use case),
- *service* ("sub" use-case), and
- *message* (class-level functions or methods, also called object behavior)

Often the term *behavior* covers all this, and telecom people talk of service as the user-level behavior, not the "middle-level" abstraction -- cf. quality-of-service. With NTNU's mixed crowd of computer science and telematics students, there can easily be confusion.

Conditions and constraints are not used in a clear way. E.g. a state or sequence diagram usually has *conditions* on the transitions and actions, not constraints.

D. Defect and observer forms

Merge the 14 forms into two, with 7 in each document, since each team have to use 4 of 7 forms anyhow as part of their final report.

Defect forms: uses both the terms *discrepancy* and *defect.*

E. OORT reading instructions

See the changed NTNU version vs. the original UMD version in Appendix F, with explanations below.

Should name them e.g. OORT-1, not only Reading 1 etc.

Sub-techniques should be named Step R1.1, R1.2 etc.

In general: very wordy and with too much boldface print, so I suggest a more operational version, with more explicit bullets and questions, e.g. named Q1.2.a, Q1.2b etc. (see below). This also makes cross-refs from defect reports easier, and we can perhaps find the most "efficient" techniques and questions.

The slide with vertical/horizontal techniques has misplaced the horizontal OORT-2 link (see below).

Functional Requirements => Requirements Description (RD), since they contain both functional and non-functional requirements.

Abbreviations: State Diagram (StD, not only SD),
Sequence Diagram (SqD to distinguish from SD),
Class Diagram (CD), Class Description (CDe).

Wrt. input/outputs of sub-techniques: May sometimes need to specify inputs as SET OF ..., and class descriptions were missing as explicit inputs. Also, sometimes the outputs are stated as marked (annotated) states and actions, other times their associated diagrams are also stated.

Suggest *singular* forms all over: e.g. Class Diagram x Class Description.

Omission defect category: discuss more whether it is vertical or horizontal. In a case where one design artifact contains "more" information (e.g. a class or a message) than the other, which artifact "wins"? -- this may be both *omission* or *extraneous fact* and we may have to consult non-design documents. However, this is mostly referred to as *omission* in the OORT guidelines.

Add severity to the defect classification. e.g. *minor, major, supermajor*?

Below follows a list of more major comments to the seven OORTs (aside from shortening them and being more precise about inputs and outputs):

OORT-1 Reading 1: Sequence Diagram x Class Diagram

OORT-2 Reading 2: State Diagram x Class Description

As mentioned: Wrongly indicated as State Diagram x Class Diagram on the figure showing vertical/horizontal connections in the lecture slides.
R2.1: Type of object should be called the *class* of the object.
R2.2: Removed blue-marked attributes and green-marked behaviors as outputs, as these are not being used later (according to Forrest, this should be done also in the UMD 735 version).

OORT-3 Reading 3: Sequence Diagram x State Diagram

OORT-4 Reading 4: Class Diagram x Class Description

See actual changes in Appendix D.

OORT-5 Reading 5: Class Description x Requirement Description

OORT-6 Reading 6: Sequence diagram x Use case

The conditions/constraints miss a marking color, so what about "double-green"?

OORT-7 Reading 7: State Diagram x (Requirement Description and Use Case)

The vertical inputs should be extended to also include Use cases.

F. File versions and naming

Presently, the naming and versioning is rather chaotic. See recommended list in section 4.3.

For instance, I have 4 correct requirements and 3 seeded ones on LA:

Ex. LA_req-good.doc, LA-req.doc,
LA_Requirements.doc, LA_Requirements (1).doc,

Ex. LA_req-bad.doc, LA_ReqSeeded.doc, LA_Req_seeded.doc.