|  |
| --- |
|  |
| Software Engineering |
| Part II. Software Requirements Analysis |

|  |
| --- |
| Dr. Moghadampour, Ghodrat  Spring 2023 |

**Contents**

1. [Software Requirements Analysis](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#SRA)
   1. [Requirements Elicitation](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#ReEl)
      1. [Initiating the Process](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#ITP)
      2. [Facilitated Application Specification Techniques](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#FAST)
      3. [Quality Function Deployment](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#QFD)
   2. [Analysis Principles](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#AP)
      1. [The Information Domain](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#TID)
      2. [Modelling](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Mod)
      3. [Partitioning](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Par)
      4. [Essential and Implementation Views](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#EAIV)
   3. [Software Prototyping](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#SP)
      1. [Prototyping Approaches](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#PA)
      2. [Prototyping Methods and Tools](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#PMAT)
   4. [Object-Oriented Analysis](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#OOA)

**1.2 Software Requirements Analysis**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**Software requirements engineering** is a set of activities, which is also called analysis. It is **a process of discovery, refinement, modelling and specification**. The system requirements and role allocated to software are refined in detail.

* **Model of the required data, information and operational behaviour are created**.
* **Alternative solutions are analysed and a complete analysis model is created**.
* **Both the software engineer and customer take an active role in software requirement engineering**.
* **The customer tries to formulate his understanding of the system, its functions and behaviour into concrete detail. The developer acts as interrogator, consultant, problem solver and negotiator**.

In requirements analysis and specification **communication content is high**. **Chances for misinterpretation or misinformation are very high. Ambiguity is probable.** The customer may say something like this to system developer: "I know you believe you understood what you think I said, but I am not sure you realize that what you heard is not what I meant?"

**1.2.1 Requirements Analysis**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**Requirements analysis is a software engineering task that bridges the gap between system level requirements engineering and software design.**  
  
**Requirements engineering activities result in the specification of software's operational characteristics, i.e. function, data and performance, indicate software's interface with other system elements and establish constraints that software must meet.**  
  
Software requirements analysis may be divided into five areas of effort:

* **Problem recognition**
* **Evaluation and synthesis**
* **Modelling**
* **Specification**
* **Review**

Initially, **the analyst studies the system specification, if one exists and the software project plan**. The software must be understood in a system context and the **software scope, which was used to generate planning estimates must be reviewed**.  
  
Next, communication for analysis must be established so that **problem recognition is ensured**. The goal is **recognition of the basic problem elements as perceived by the customer and users**.  
  
**Problem evaluation and solution synthesis is the next major areas of effort for analysis**. At this stage **the goal is to describe the problem so that an overall approach or solution may be synthesized**.  
  
The analyst must define:

* **All externally observable data objects**
* **Evaluate the flow and content of information**
* **Define and elaborate all software functions**
* **Understand software behaviour in the context of events that affect the system**
* **Establish system interface characteristics, and uncover additional design constraints**

For instance, a room reservation system is required for a hotel. The problems an analyst may find with the current manual system may include: 1) the inability to obtain the status of a room rapidly, 2) two or three days turnaround to update a card file, 3) multiple assignments of the same room to different customers, etc.  
  
**Once problems have been identified, the analyst determines what information is to be produced by the new system and what data will be provided to the system**. For example, the customer desires a daily report of occupied rooms with the information of their occupants.  
  
**Upon evaluating current problems and desired information, i.e. input and output the analyst begins to synthesize one or more solutions**.  
  
To begin, **the data objects, processing functions and behaviour of the system are defined in detail**. Once this information has been established, **basic architectures for implementation are considered**.  
  
A client/server approach may seem to be appropriate, but does the software to support this architecture fall within the scope outlined in the software plan? A database management system would seem to be required, but is the user/customer's need for associativity justified?  
  
**The process of evaluation and synthesis continues until both analyst and customer feel confident that software can be adequately specified for subsequent development steps**.  
  
**Throughout the evaluation and solution synthesis, the analyst's primary focus is on what not how**:

* What data does the system produce and consume
* What functions must the system perform, what behaviour does the system exhibit
* What interfaces are defined and what constraints apply?

**During the evaluation and solution synthesis activity**, the **analyst creates models of the system** in an effort **to better understand data and control flow, functional processing, operational behaviour and information content**.  
  
**The model serves as a foundation for software design and the basis for the creation of a specification for the software**.  
  
**The detailed specification may not even be possible at this stage**.  
  
**The customer may be unsure of precisely what is required.**  
  
**The developer may be unsure that a specific approach will properly accomplish function and performance**.  
  
**For these and other similar reasons an alternative approach to requirements analysis, called prototyping may be conducted**.

**1.2.2 Requirements Elicitation**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
Before requirements can be analysed, modelled or specified  
  
they must be gathered through an elicitation process. A customer has a problem that may be amenable to a computer-based solution. A developer responds to the customer's request for help. Communication has begun, but the road from communication to understanding is often full of obstacles.

**1.2.2.1 Initiating the Process**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**The most commonly used requirements elicitation technique is to conduct a meeting or interview**.  
  
**In the first meeting between a software engineer or the analyst and the customer, neither might know what to say or ask**; both might be worried that what they do say will be misinterpreted; both are thinking about where it might lead and both likely have radically different expectations.  
  
**The analyst can start by asking context free questions**. That is a set of questions that will lead to a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of the first encounter itself.  
  
**The first set of context free questions focus on the customer, the overall goals and benefits**. For example, the analyst might ask:

* **Who is behind the request for this work?**
* **Who will use the solution?**
* **What will be the economic benefit of a successful solution?**
* **Is there another source for the solution that you need?**

These questions help to identify all stakeholders who will have an interest in the software to be built. In addition, **the questions identify the measurable benefit of a successful implementation and possible alternatives to custom software development**. The n**ext set of questions enable the analyst to gain a better understanding of the problem and the customer to voice her perception about a solution**:

* **How would you characterize good output that would be generated by a successful solution?**
* **What problems will this solution address?**
* **Can you show me or describe the environment in which the solution will be used?**
* **Are there special performance issues or constraints that will affect the way the solution is approached?**

**The final set of questions focus on the effectiveness of the meeting**. These meta-questions are like the followings:

* **Are you right person to answer these questions? Are your answers official?**
* **Are my questions relevant to the problem that you have?**
* **Am I asking too many questions?**
* **Is there anyone else who can provide additional information?**
* **Is there anything else that I should be asking you?**

**These and other questions will help to break the ice and initiate the communication that is essential to successful analysis**. But a question and answer meeting format is not an approach that has been overwhelmingly successful. **The Q&A session should be used for the first encounter only and then be replaced by a meeting format that combines elements of problem solving, negotiation, and specification**.

**1.2.2.2 Facilitated Application Specification Techniques**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**Too often, customers and software engineers have an unconscious "us and them" mind set**. **Rather than working as a team to identify and refine requirements, each constituency defines its own territory and communicate through a series of memos, formal position papers, documents and question and answer sessions**. Experience has shown that this approach doesn't work very well. Misunderstanding is a lot, important information is omitted and a successful working relationship is never established.  
  
Due to problems mentioned above some independent investigators have developed **facilitated application specification techniques (FAST), which is a team-oriented approach to requirements gathering and is applied during early stages of analysis and specification**. FAST encourages **the creation of a joint team of customers and developers who work together** to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements.  
  
All approaches to FAST apply some variation of the following basic guidelines:

* **A meeting is conducted at a neutral site and attended by both software engineers and customers**
* **Rules for preparation and participation are established**
* **An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas**
* **A facilitator, that can be a customer, a developer or an outsider controls the meeting**
* **A definition mechanism, which can be work sheets, flip charts, wall stickers or an electronic bulletin board, chat room or virtual forum is used**
* **The goal is to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements in an atmosphere that is conductive to the accomplishment of the goal.**

As the FAST meeting begins, the first topic of discussion is the need and justification for the new product; everyone should agree that the product is justified.  
  
Once agreement has been established, each participant presents his or her list for discussion.  
  
The lists can be pinned to the walls of the room using large sheets of paper, stuck to the walls using adhesive backed sheets or written on a wall board.  
  
Alternatively, the lists may have been posted on an electronic bulletin board or posted in a chat room environment for review prior to the meeting.  
  
Ideally, each list entry should be capable of being manipulated separately so that lists can be combined, entries can be deleted and additions can be made. At this stage, critique and debate are strictly prohibited.  
  
**After individual lists are presented in one topic area, a combined list is created by the group**. The combined list eliminates redundant entries, adds any new ideas that come up during the discussion, but does not delete anything.  
  
After combined lists for all topic areas have been created, discussion coordinated by facilitator ensues. The combined list is shortened, lengthened, or reworded to properly reflect the product/system to be developed.  
  
**The objective is to develop a consensus list in each topic area; objects, services, constraints and performance**. The lists are then set aside for later action.  
  
Once the consensus lists have been completed, the team is divided into smaller sub-teams; each works to develop a mini-specification for one or more entries on each of the lists. An alternative approach results in the creation of use cases.  
  
The **mini-specification** is an elaboration of the word or phrase contained on a list. For example, the mini-specification for the RoomReserveationSystem object room might be: it is of two types: smoking and non-smoking, and three sizes: single bed, double bed, or family, it can be in three status: vacant, reserved or occupied and the authorized person can modify its status; assign it to a customer or free it.  
  
Each sub-team then presents each of its mini-specs to all FAST attendees for discussion. Additions, deletions and further elaboration are made. In some cases, the development of mini-specs will uncover new objects, services, constraints or performance requirements that will be added to the original lists. During all discussions, the team may raise an issue that cannot be resolved during the meeting. An issues list is maintained so that these ideas will be acted on later.  
  
After the mini-specs are completed, each FAST attendee makes a list of **validation criteria** for the product/system and presents his or her list to the team. A consensus list of validation criteria is then created. Finally, one or more participants or outsiders is assigned the task of writing the complete draft specification using all inputs from the FAST meeting.  
  
FAST is not a panacea for the problems encountered in early requirements elicitation. But **the team approach provides the benefits of many points of view, instantaneous discussion and refinement, and is a concrete step toward the development of a specification**.

**1.2.2.3 Quality Function Deployment**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**Quality function deployment (QFD) is quality management technique that translates the needs of the customer into technical requirements for software**.  
  
**QFD concentrates on maximizing customer satisfaction from the software engineering process.**  
  
**QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process**.  
  
QFD identifies three types of requirements:

* **Normal requirements (must have)**: The objectives and goals that are stated for a product or system during meetings with the customer. **If these requirements are present, the customer is satisfied**. Examples o normal requirements might be requested types of graphical displays, specific system functions and defined levels of performance.
* **Expected requirements (should have)**: These requirements are implicit to the product or system and **may be so fundamental that the customer does not explicitly state them**. **Their absence will be a cause for significant dissatisfaction**. Examples of expected requirements are: ease of human-machine interaction, overall operational correctness and reliability, and ease of software installation.
* **Exciting requirements (nice to have)**: These features **go beyond the customer's expectation's and prove to be very satisfying when present**. For example, word processing software is requested with standard features. The delivered product contains a number of page layout capabilities that are quite pleasing and unexpected.

QFD spans the entire engineering process. An overview of these concepts adapted for computer software is:

* In meetings with the customer, function deployment is used to determine the value of each function that is required for the system.
* Information deployment identifies both the data objects and events that the system must consume and produce. These are tied to the functions.
* Task deployment examines the behaviour of the system or product within the context of its environment.
* Value analysis is conducted to determine the relative priority of requirements determined during each of the three deployments noted above.

QFD uses customer interviews and observation, surveys and examination of historical data, e.g. problem reports as raw data for the requirements gathering activity. These data are then translated into a table of requirements, called the customer voice table, that is reviewed with the customer. A variety of diagrams, matrices and evaluation methods are then used to extract expected requirements and to attempt to derive exciting requirements.

**1.3 Analysis Principles**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
A large number of analysis modelling methods have been developed. Many problems related to analysis have been identified and a variety of modelling notations and corresponding sets of heuristics to overcome them have been developed.  
**All analysis methods are related by a set of operational principles:**

* **The information domain of a problem must be represented and understood.**
* **The functions that the software is to perform must be defined.**
* **The behaviour of the software as a consequence of external events must be represented.**
* **The models that depict information, function and behaviour must be partitioned in a manner that uncovers detail in a layered or hierarchical fashion.**
* **The analysis process should move from essential information toward implementation detail.**

By applying these principles the analyst approaches a problem systematically. The **information domain** is examined so that function may be understood more completely. **Models** are used so that the characteristics of function and behaviour can be communicated in a compact fashion. **Partitioning** is applied to reduce complexity. **Essential** and **implementation views** of the software are necessary to accommodate the logical constraints imposed by processing requirements and the physical constraints imposed by other system elements.  
  
In addition to the operational analysis principles the following guiding principles for requirements gathering might be used:

* **Understand the problem before you begin to create the analysis model**. There is a tendency to rush to a solution, even before the problem is understood. This often leads to elegant software that solves the wrong problem.
* **Develop prototypes that enable a user to understand how human-machine interaction will occur**. Since the perception of the quality of software is often based on the perception of the friendliness of the interface, prototyping and the iteration that results are highly recommended.
* **Record the origin of and the reason for every requirement**. This is the first step in establishing traceability back to the customer.
* **Use multiple views of requirements**. Building data, functional and behavioural models provide the software engineer with three different views. This reduces the likelihood that something will be missed and increases the likelihood that inconsistency will be recognized.
* **Prioritize requirements**. Tight deadlines may preclude the implementation of every software requirement. The most important requirements must be separated from less important ones.
* **Work to eliminate ambiguity**. Because most requirements are described in a natural language, the opportunity for ambiguity is very high. The use of formal technical review is one way to uncover and eliminate ambiguity.

**1.3.1 The Information Domain**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**All software applications process data. They transform data from one form to another; they accept input, manipulate it in some way and produce output**.  
  
**Software also processes events**. An event represents some aspect of system control and is realty nothing more than Boolean data; it is either on or off, true or false, there or not there. For example, in a fire detection system, a fire detector detects fire and sends an alarm signal to monitoring software. The alarm signal is an event that controls the behaviour of the system. Therefore, data, e.g. numbers, text, images, sounds, video, etc. and control (events) both reside within the information domain of a problem.  
  
The first operational analysis principle requires an examination of the information domain and the creation of a data model. The information domain contains three different views of the data and control as each is processed by a computer program:

1. information content and relationships, i.e. the data model
2. information flow
3. information structure

**Information content** represents the individual data and control objects that comprise some larger collection of information that is transformed by the software. For example, the data object, customer, in a RoomReservation application is a composite of a number of important pieces of data: customer�s name, room number, and duration of stay. Therefore, the content of customer is defined by the attributes that are needed to create it.  
  
Data and control objects can be related to other data and control objects. For example, the data object customer has one or more relationships with objects room, payment and others. During the analysis of the information domain these relationships should be defined.  
  
**Information flow** represents the manner in which data and control changes as each moves through a system. Objects are transformed to intermediate information, e.g. data and/or control, that is further transformed to output. Along this transformation path additional information may be introduced from an existing data store, e.g. disk file or memory buffer. The transformations that are applied to the data are functions or subfunctions that a program must perform. Data and control that move between two transformations (functions) define the interface for each function.  
  
**Information structure** represents the internal organization of various data and control items. Are data or control items to be organized as an n-dimensional table or as a hierarchical tree structure? Within the context of the structure, what information is related to other information? Is all information contained within a single structure or are distinct structures to be used? How does information in one information structure relate to information in another structure? These questions and others are answered by an assessment of information structure. The data structure refers to the design and implementation of information structure within the software.

**1.3.2 Modelling**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**We create functional models to gain a better understanding of the actual entity to be built**. When the entity is a physical thing, e.g. a car, a room, etc. we can build a model that is identical in form and shape, but smaller in scale. However, when the entity to be built is software, our model must take a different form. It must be capable of representing the information that software transforms, the functions and subfunctions that enable the transformation to occur, and the behaviour of the system as the transformation is taking place.  
  
The second and third operational analysis principles require that we build models for function and behaviour.

* **Functional models**: Software transforms information and in order to accomplish this, it must perform at least three generic functions: input, processing and output. When functional models of an application are created, the software engineer focuses on problem specific functions. The functional model begins with a single context level model, i.e. the name of the software to be built. Over a series of iterations, more and more functional detail is provided, until a thorough description of all system functionality is represented.
* **Behavioural models**: Most software responds to events from the outside world. This stimulus-response characteristic forms the basis of the behavioural model. A computer program always exists in some state, which is an externally observable mode of behaviour, e.g. loading, saving, printing, and is changed only when some event occurs. For example, software will remain in the wait state until 1) an internal clock indicates that some time interval has passed or 2) an external event like mouse press causes an interrupt. A behavioural model creates a representation of the states of the software and the events that cause a software to change state.

Models created during requirements engineering serve a number of important roles:

* The model aids the analyst in understanding the information, function and behaviour of a system, thereby making the requirements analysis task easier and more systematic
* The model becomes the focal point of review, and therefore the key to a determination of completeness, consistency and accuracy of the specification
* The model becomes the foundation for design, providing the designer with an essential representation of software that can be mapped into an implementation context.

**1.3.3. Partitioning**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**Problems are often too large and complex to be understood as a whole. For this reason, we tend to a divide such problems into parts that can be easily understood and establish interfaces between the parts so that overall function can be accomplished**. The fourth operational analysis principle suggests that the information, functional and behavioural domains of software can be partitioned.  
  
In essence, partitioning decomposes a problem into its constitute parts. Conceptually we establish a hierarchical representation of function or information and then partition the uppermost element by:

1. exposing increasing detail by moving vertically in the hierarchy or
2. functionally decomposing the problem by moving horizontally in the hierarchy

**1.3.4 Essential and Implementation Views**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**An essential view of software requirements presents the functions to be accomplished and information to be processed without regard to implementation details**. For example, the essential view of the RoomReservation function reserve room does not concern itself with the physical form of the data. Similarly an essential data model of the data item room number used by the function *reserveRoom()* can be presented at this stage without regard to the underlying data structure used to implement the data item. By focusing attention on the essence of the problem at early stages of requirements engineering, we leave our options open to specify implementation details during later stages of requirements specification and software design.  
  
**The implementation view of software requirements presents the real world manifestation of processing functions and information structures**. In some cases, a physical representation is developed as the first step in software design. However, most computer-based systems are specified in a manner that dictates accommodation of certain implementation details. The general characteristics of the hardware and software parts of the system should be noted as part of a software requirements specification. The analyst must recognize the constraints imposed by pre-defined system elements, e.g. the database management system, and consider the implementation view of function and information when such a view is appropriate.  
  
It should be noted that software requirements engineering focuses on what the software is to accomplish, rather than on how processing will be implemented. However, the implementation view should not necessarily be interpreted as a representation of how. Rather, an implementation model represents the current mode of operation, that is, the existing or proposed allocation for all system elements. The essential models of function and data are generic in the sense that realization of function is not explicitly indicated.

**1.4 Software Prototyping**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
An Analysis should be conducted regardless of the software engineering paradigm that is applied. However, the form that analysis takes will vary. In some cases **it is possible to apply operational analysis principles and derive a model of software from which a design can be developed**.  
  
**In other situations, requirements elicitation via FAST, QFD, use-cases or other brainstorming techniques is considered, analysis principles are applied and a model of the software to be built, called a prototype is constructed for customer and developer assessment**.  
  
Finally, **there are circumstances that require the construction of a prototype at the beginning of analysis, since the model is the only means through which requirements can be effectively derived**. The model then evolves into production software.

**1.4.1 Prototyping Approaches**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
The prototyping paradigm can be either **close-ended** or **open-ended**.

* **The close-ended approach is often called throwaway prototyping**. Using this approach, a prototype serves solely as a rough demonstration of requirements. It is then discarded and the software is engineered using a different paradigm.
* **An open-ended approach, called evolutionary prototyping**, uses the prototype as the first part of an analysis activity that will be continued into design and construction. The prototype of the software is the first evolution of the finished system.

**Before a close-ended or open-ended approach can be chosen, it is necessary to determine whether the system to be built is amenable to prototyping**. In general, any application that creates dynamic visual displays, interacts heavily with a user, or demands algorithms or combinatorial processing that must be developed in an evolutionary fashion is a candidate for prototyping. However, these application areas must be weighed against application complexity.  
  
If a candidate application will require the development of tens of thousands of lines of code before any demonstrable function can be performed, it is likely to be too complex for prototyping. If, however, the complexity can be partitioned, it may still be possible to prototype options of the software.  
  
**Because the customer must interact with the prototype in later steps, it is essential that:**

* **Customer resources be committed to the evaluation and refinement of the prototype, and**
* **The customer is capable of making requirements decisions in a timely fashion.**

Finally, **the nature of the development project will have a strong bearing on the efficacy of prototyping**:

* **Is project management willing and able to work with the prototyping method?**
* **Are prototyping tools available?**
* **Do developers have experience with prototyping methods?**

**1.4.2 Prototyping Methods and Tools**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
For software prototyping to be effective a prototype must be developed rapidly so that the customer may assess results and recommend changes. To conduct rapid prototyping, three generic methods and tools are available:

* **Fourth Generation Techniques**: Fourth Generation Techniques (4GT) encompass a broad array of database query and reporting language, program and application generators and other very high level non-procedural languages. Because 4GT enable the software engineer to generate executable code quickly, they are ideal for rapid prototyping.
* **Reusable Software Components**: Another approach to rapid prototyping is to assemble, rather than build, the prototype by using a set of existing software components. Melding prototyping and program component reuse will work only if a library system is developed so that components that do exist can be catalogued and then retrieved. It should be noted that an existing software product can be used as prototype for a new improved competitive product. In a way, this is a form of reusability for software prototyping.
* **Formal Specification and Prototyping Environments**: A number of formal specification languages and tools have been developed as a replacement for natural language specification techniques. Today, developers of these formal languages are in the process of developing interactive environments that 1) enable an analyst to interactively create a language-based specification of a system or software, 2) invoke automated tools that translate the language-based specification into executable code and 3) enable the customer to use the prototype executable code to refine formal requirements.

**1.5 Object-Oriented Analysis**

[Contents](https://portal.vamk.fi/pluginfile.php/341752/mod_resource/content/7/Lectures/software_requirements_analysis.html#Con)  
**Structured analysis** takes a distinct input->process->output view of requirements. Data are considered separately from the processes that transform the data. System behavior, although important, tends to play a secondary role in structured analysis. The structured analysis approach makes heavy use of functional decomposition (partitioning of the data flow diagram).  
  
In **object oriented analysis** system is thought of beign composed of classes and their instances, i.e. objects. OOA is based upon concepts such as objects and attributes, classes and members, wholes and parts.  
  
In OOA we try to characterize system according to object oriented software engineering by answering a set of questions. By trying to answer the following questions we model the system in an object oriented way. Each of these questions is answered within the context of object-oriented analysis (OOA), which is naturally the first technical activity performed as part of OO software engineering.

* What are the relevant objects?
* How do they relate to one another?
* How do objects behave in the context of the system?
* How do we specify or model a problem so that we can create an effective design?

In order to build an analysis model, five basic principles are applied:

1. The information domain is modeled.
2. Module function is described.
3. Model behavior is represented.
4. The models are partitioned to expose greater detail.
5. Early models represent the essence of the problem while later models provide implementation details.

The intent of OOA is to define all classes (and the relationships and behavior associated with them) that are relevant to the problem to be solved. To accomplish this, a number of tasks must occur:

1. Basic user requirements must be communicated between the customer and the software engineer.
2. Classes must be identified (i.e., attributes and methods are defined).
3. A class hierarchy must be specified.
4. Object-to-object relationships (object connections) should be represented.
5. Object behavior must be modeled.
6. Tasks 1 through 5 are applied iteratively until the model is complete.

Instead of examining a problem using the classic input->processing->output(information flow) model or a model derived exclusively from hierarchical information structures, OOA introduces a number of new concepts.  
  
The objective OOA is to develop a series of models that describe computer software as it works to satisfy a set of customer defined requirements. OOA, like the conventional analysis methods, builds a multipart analysis model to satisfy this objective. The analysis model depicts information, function, and behavior within the context of the elements of the object model.