Non-Functional Requirements

(based on materials from Mei Nagappan, Taylor et al., 2010)
Motivation – Contract Negotiation Table

Service Provider’s Team

Client’s Team

Stakeholders
Stakeholder?

“A person or organization that has rights, share, claims or interests in the system under construction or its properties satisfying their needs and expectation.”

(Nikolay Ashanin, 2017)

More on stakeholders!

- Tom Gib says - “there is always one or more System stakeholder than you know about; and the known stakeholders have at least one more need than you know about now.
- Stakeholders use the architectural documents to understand and approve the system under construction.
Categories of Stakeholders?

- Developers
- Managers
- QA
- System Under Construction
- Maintenance
- Customers
- Sales
- DevOps
- Support

Opeyemi Adesina - Software Design and Architecture - Spring 2018
Stakeholders’ Questions.

- **Management** – are we on **schedule**?
- **Developers** – who is responsible for what?
- **Sales** – can we claim it can do the task?
- **Quality Assurance Specialists** – what **team** do we talk to about **defects**?
- **DevOps** – where should this **component be deployed**?
- **Support** – which **QA team signed off** on this?
- **Maintenance** – how can we add this **feature**?
How the customer explained it
How the project leader understood it
How the programmer wrote it
How the analyst designed it
How the business consultant described it
What the customer really needed

Source: http://projectcartoon.com
Conflicts among Stakeholders.

- Categories of system properties:
  - Functional properties – what the system is supposed to do ("the system shall do X").
  - Non-Functional properties (NFPs) – what the system is supposed to be ("the system shall be Y").

- Stakeholders might have conflicting opinions about what NFRs matter most:
  - e.g., development team cares about maintainability more than the customers.
  - e.g., QA will be more interested in testability of the application than sales.
Resolving Conflicts – Tradeoffs.

Client (Customer, Sponsor)
- Low cost
- Increased Productivity
- Backward-Compatibility
- Traceability of requirements
- Rapid development
- Flexibility

End User
- Functionality
- User-friendliness
- Ease of Use
- Ease of learning
- Fault tolerant
- Robustness

Developer/Maintainer
- Minimum # of errors
- Modifiability, Readability
- Reusability, Adaptability
- Well-defined interfaces

Runtime Efficiency
Reliability
Portability
Good Documentation
Examples of possible tradeoffs.

- Functionality vs. Usability
- Cost vs. Robustness
- Efficiency vs. Portability
- Development Velocity vs. Functionality
- Cost vs. Reusability
- Backward Compatibility vs. Readability
Non-Functional Properties (NFPs)

NFPs are constraints on the manner in which the system implements and delivers its functionality.

- **Efficiency** – focuses on resource utilization and economy.
- **Complexity** – degree of difficulty in understanding or verifying its component(s).
- **Heterogeneity** – quality integration of diverse components in different environments.
- **Adaptability** – meeting new requirements and operating conditions in its lifetime.
- **Scalability** – ability to adapt to meeting new requirements of size and scope.
- **Dependability** – composed from other NFPs (i.e., reliability, availability, robustness, fault-tolerance, survivability, safety, and security).
Distinguishing NFPs from FPs
Non-Functional Properties (contd.)

NFPs play a critical role in how the product is perceived by end-users.

- “It’s damn slow.”
- “The app keeps crashing.”
- “It doesn’t work with my [OS, …]”
- “the pictures are very clear.”
- “this app is super-cool.”
- Products are sold based on their FPs.
Design Guidelines for NFPs

- Provide guidelines that support various NFPs.

- **How to?** – Focus on architectural levels:
  - Components,
  - Connectors, and
  - Topologies.
Evaluating NFPs

- Don’t be tempted to treat NFPs as abstract entities.

- Concrete treatment for NFPs – thinking about how they can be measured.

- If you do not do this, it is hard to validate whether a design/architectural decision supports or inhibits an NFP.
NFP - Efficiency

- Efficiency is a quality that reflects a system’s ability to meet its performance requirements.

Components:
- Keep them small.
- Simple and compact interfaces.
- Allow multiple interfaces to the same functionality
- Separate data from processing components.
- Separate data from meta data.

Connectors:
- Carefully choose connectors.
- Be careful of broadcast connectors.
- Encourage asynchronous interaction.
- Be wary of location/distribution transparency.

Topology:
- Keep frequent collaborators “close”.
- Consider efficiency impact of selected styles.
 Complexity is a property that is proportional to the size of a system, its volume of constituent elements, their internal structure, and their interdependencies.

**Components:**
- Separate concerns into components.
- Keep functionality in components - not interaction.
- Ensure cohesiveness (keep it small).
- Insulate processing from data format changes.
- Be aware of impacts of off-the-self components.

**Connectors:**
- Keep only interaction in connectors.
- Restrict interactions supported by each connector. (e.g., msg. broadcast)
- Separate interaction concerns into connectors.

**Topology:**
- Eliminate unnecessary dependencies.
- Manage all dependencies explicitly.
- Use hierarchical (de)composition.
NFP – Scalability and Heterogeneity

- **Scalability** – the capability of a system to adapt to meet new size and scope requirements.

- **Heterogeneity** – A system’s ability to be composed of, or execute within disparate parts.

- **Portability** – A system’s ability to execute on multiple platforms with minimal modifications, while retaining its functional and non-functional properties.
NFP – Scalability

Components:
- handle **single and clearly defined** task.
- have **simple and understandable** interface.
- **not be delegated** to undertake responsibilities of interactions.
- avoid unnecessary heterogeneity.
- data sources can be distributed across components.
- replicate data (where necessary).

Connectors:
- Use explicit connectors.
- Give each connector a clearly defined responsibility.
- Choose the simplest connector suited for the task.
- Beware of differences btw (in)direct dependencies.
- Maximize explicit connectors to support data scalability (e.g., caching, prefetching, buffering)

Topology:
- Avoid system bottlenecks (load balancing).
- Utilize parallel processing capabilities.
- Place data sources near data consumers.
- Make distribution locationally transparent.
Evolvability – ability to change to satisfy new requirements and environments.

Components:
- Same as for complexity but the goal is to reduce risks by isolating modifications.

Connectors:
- Clearly define responsibilities.
- Make connectors flexible.

Topology:
- Avoid implicit connectors.
- Encourage location transparency.
NFP – Dependability (Tailor et al., 2010)

- **Reliability** – probability that a system will perform within its design limits without failure over time.
- **Availability** – probability that a system is accessible at a particular instant in time.
- **Robustness** – ability of a system to respond adequately to unanticipated runtime conditions.
- **Fault-tolerance** – ability to respond gracefully to failures at runtime (e.g., environmental, components, connectors, component-connector mismatches).
- **Survivability** – ability to resist, recover and adapt to mission-compromising threats.
  - Sources of threats: attacks, failures and accident.
  - Activities: resist, recognize, recover, adapt.
NFP – Dependability (Tailor et al., 2010)

- **Safety** – ability to avoid failures that will cause loss of life, injury or loss of/to property.

**Components:**
- Control external components dependencies.
- Support reflection.
- Support exception handling.
- Specify the component’s key state invariants.

**Connectors:**
- Employ components that strictly control component dependencies.
- Provide appropriate component interaction guarantees.
- Support dependability techniques via advanced connectors.

**Topology:**
- Avoid single point of failure.
- Backup critical data and functionalities.
- Support system health monitoring.
- Support dynamic adaptation.