cs121 - software development requirements

alexandre r.j. francois
visiting associate professor of computer science
what are requirements?
why requirements?
software requirements specification

engineering requirements
use cases
requirements matrix
recommendations
from design/idea to software system
from design/idea to software system

what the customer really wants
what are requirements?

concisely worded statement that describes how the software will behave when it is complete

everytime a requirement is implemented, the system *shall* behave in accordance to the conditions that the requirements establish

• **functional requirements** address the operations that the system performs (behavior).

• **nonfunctional requirements** apply to the standards or qualities of performance that constrain the design or operations of the system
what are requirements?

requirements (what) are not expressed in the language of implementation (what)

  functionality of the system addressed without reference to implementation: external, or user-oriented perspective.

  example: “the user shall click a dialog button to open a new window” vs. “after the user selects the next level option, the next level appears.”

requirements engineering:

  translate features into functionality

  features come from a variety of sources (many people, some documents created by others)
types of requirements
FURPS+

**functional**: features, capabilities

**usability**: human factors, aesthetics, consistency, documentation

**reliability**: frequency of failure, recoverability, predictability

**performance**: response times, throughput, accuracy, availability, resource usage

**supportability**: adaptability, maintainability, configurability

+: others...
why requirements?

ingengineering involves stating and solving a problem; requirements is how the problem is stated
(engineer vs. hacker/performance artist)

**solving the problem**: problem must be clearly and precisely stated

**planning**: without precise specification of all behaviors, impossible to evaluate the time needed to implement the system

**testing**: without complete list of behaviors to be supported, impossible to test whether a system implemented actually behaves as it is supposed to

and: extension, cost, maintainability, process improvement

© arf 2010
why requirements?

mistakes/failures at the requirements stage are very costly!
mistakes/failures at the requirements stage are surprisingly common!!
requirements challenges

customers can’t tell you what they need
  they don’t know
  they don’t clearly articulate their needs

customers have conflicting needs
  cost vs. quality vs. time

customers' needs change

technology changes
software requirements specification

software requirements specification (SRS) document:
list of formally stated requirements
may also combine other approaches to requirements specification (e.g. in case-driven specification)

IEEE provides a document template for SRS (IEEE standard 830), that serves as a starting point to be tailored to a specific project...
table of contents

introduction
  purpose
  scope (general constraints)
  definitions, acronyms and abbreviations
  references
  overview (diagram, subsystems organization and interactions)
srs document (template)
overall description (optional)

product perspective (e.g. refer to design document, describe genre of the game, brief description of the game mechanics)

product functions (break down into different subsystems-or modules)

user characteristics (e.g. profile the player of the game)

constraints, assumptions and dependencies (e.g. DirectX, OpenGL, Win32, etc.)
external interface requirements

user interfaces: keyboard, mouse, joystick, gamepad, etc.; screens and menus, possibly from design document

hardware interfaces: modem, joystick, audio, video, ...

software interfaces: software dependencies: OS, libraries, ...

communications interfaces (e.g. modem)
srs document (template)
specific requirements

**functional requirements**

*break down requirements according to subsystems or components to which they apply:*

subsystem A
  (state requirements, number each requirement)
subsystem B
...

© arj 2010
srs document (template)
specific requirements: other requirements

performance requirements:
  standards (for specific markets)
  hardware limitations

design constraints
  availability (for networked/distributed systems)
  security
  maintainability (how to upgrade, fix, etc.)

other requirements
srs document (template)
appendices and index

appendices (additional documents)
  test cases
  requirements matrix

index
writing requirements

state requirement in the active voice
use the verb "shall"
each statement should pertain to only 1 feature at a time
use only two subjects: player (or user) and software (or system)
avoid wording that conveys assumptions about implementation

example:

11. software shall show player which player is active

example srs document:

www-scf.usc.edu/~csci201g/Sp2007/crosswinds-RS.pdf
requirements engineering

main delivery of the requirements engineering process is software requirements specification (SRS) document

translate features into functionality

features come from a variety of sources (many people, some documents created by others)

software requirements analyst (requirements engineering = requirements analysis):
  elicits, analyzes, specifies, verifies, and manages the functional and nonfunctional requirements of the software system
15th: Dehli, India, October 2007

**understanding requirements in the global economy**

“As software development is now part of the global economy, requirements engineering is the key bridge between the customer and supplier. Understanding and translating users’ needs into effective solutions has always been vital: however, as development is outsourced requirements have to reflect cultures and languages and local needs. Furthermore, understanding requirements becomes a collaborative activity across time and space.”
topics of interest include, but are not restricted to:

requirements elicitation, analysis, documentation, validation and verification
requirements specification languages, methods, processes, and tools
requirements management, traceability, viewpoints, prioritization, and negotiation
modeling of requirements (formal and informal), goals, and domains
prototyping, simulation, and animation
interaction between requirements and design
evolution of requirements over time, product families, and variability
relating requirements to business goals, products, architecture, and testing
social, cultural, global, personal and cognitive factors in requirements engineering
collaborative requirements engineering
domain-specific problems, experiences and solutions
requirements engineering

requirements engineering is an iterative and incremental process, so is requirements engineering approach (not a methodology) defines four increments (or phases), iterated as many times as necessary (or possible):

elicitation
exploration
analysis
refinement
requirements engineering:
elicitation

gather information about requirements from the design
document and other sources
includes prototypes, UI mockups, etc.
create a first draft of the SRS

game description:
outer scope (genre, mechanics, etc.) - high level
play narratives (sixty-second-of-play narratives):
inner scope (what it is like to play the game) - detailed
requirements engineering: exploration

create a candidate list of use cases
create an initial list of requirements
update the SRS

develop use cases:

- narratives: informal paragraph that tells a story about how the user interacts with the system
- scenarios: more formal, narrative broken down into a numbered sequence of events
- diagrams (UML)
use cases

concurrently develop use cases and list of requirements

some lines in a use case might refer to one or several more detailed use cases

for games, develop both:
  game context use cases (outer scope), and
  sixty-seconds-of-play use cases (inner scope)
use case template

use case name
requirement(s) explored
player (actor) context (role)
precondition(s)
trigger(s)
main course of action
alternate course(s) of action
exceptional course(s) of action

see examples in:

www-scf.usc.edu/~csci201g/Sp2007/crosswinds-RS.pdf
use cases uses

requirements:
  define functional requirements (behavior)
  expose business rules (constraints on behavior)
planning: suggest an iterative strategy
design: validate design models
testing: provide scenarios for validation
requirements engineering:
analysis

develop use cases based on candidate list
test initial list of requirements for completeness and validity
add use cases and requirements
(generate a test case for each use case)

subject candidate requirements to procedures that reveal weaknesses, redundancies and gaps
develop more use case diagrams, scenarios and activity diagrams

© arj 2010
requirements engineering: refinement

prioritize the requirements
create a **requirements matrix**
refine requirements language
refine test cases
refine SRS
requirements matrix

a tabular representation of the requirements headings:
\begin{itemize}
\item req number (index)
\item \textbf{status} (completion %)
\item title
\item use cases (that reference the requirement)
\item test cases (that reference the requirement)
\item dependencies
\item \textbf{priority} (1-5)
\item class references
\end{itemize}
must be kept current!
other modeling tools

use cases are not the only tool
state diagrams, activity diagrams (see UML)
prototypes, mock-ups

anything that helps!
recommendations

Care not to create incomplete, spotty, redundant, or inaccurate requirements; seek to find the essential behavior of the system and to specify this behavior in nontechnical, clearly stated terms

establish the scope of the project: boundary of the engineering effort, from design document (game overview). Inner scope vs. outer scope

identify the customers (stakeholders): end users; but also game/system designers, graphics, music and sound effects designers, voice recorders, etc. who expect that the software system will give life to what they have contributed
recommendations

feasibility: 3 major risks are
  number of features
  amount of money (resources)
  quality of the product

uncontrolled growth
  feature creep: at requirements specification time, go
    beyond established scope; at implementation time,
    developers add unspecified functionality
  goldplating: developers develop the technology they want
    to develop rather than the product the requirements
    specify
recommendations

eight basic qualities that make for good requirements specifications:

make requirements complete
make requirements correct
necessary requirements only
consider the feasibility
prioritize requirements (critical path vs. secondary)
eliminate ambiguity
verify and validate requirements
manage evolution
quality requirements

specify what not how
unambiguous
testable
feasible
consistent
prioritized
traceable
agreed upon by customer