

csc444h: software engineering I

matt medland

matt@cs.utoronto.ca

<http://www.cs.utoronto.ca/~matt/csc444>

requirements analysis

requirements analysis



quality = fitness for purpose

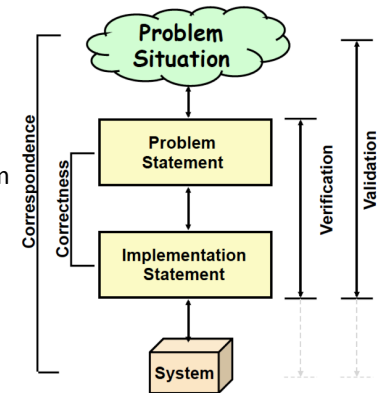
- software is everywhere
 - but our experience with it is often disappointing
- software is designed for a purpose
 - if it doesn't work well then either:
 - the designer didn't have an adequate understanding of the purpose, or
 - we are using it for something other than what it was designed for
- the purpose is found in human activities
 - ex. what do customers use bank software for?
 - different kinds of users & activities, many may be conflicting

designing for people

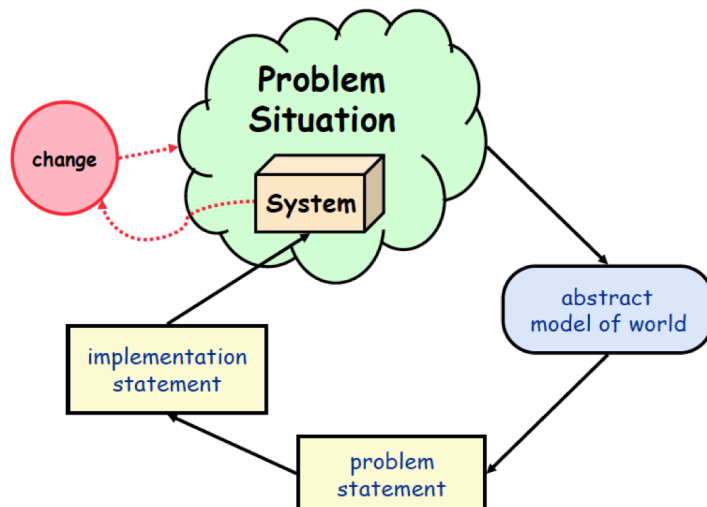
- what is the goal of the software design?
 - creating new programs, components, algos, Uis
 - making human activities more effective, efficient, safe, enjoyable
- how rational is the design process?
 - hard systems view: problems can be decomposed systematically, reqs represented formally, spec validated for correctness, correct program satisfies spec
 - soft systems view: soft dev embedded in complex org context, multiple stakeholders, different values/goals, ongoing learning process, can never adequately capture spec, participation of users is essential to process
 - reconciliation: hard systems view is ok if there is local consensus on the nature of the problem

separate problem & solution

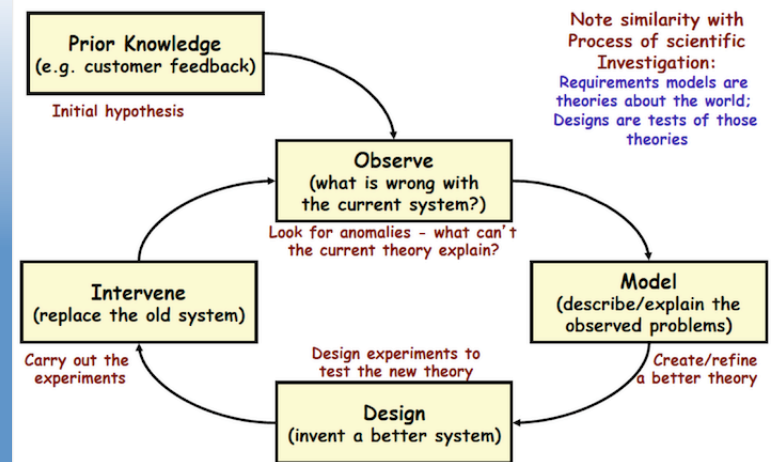
- separate problem desc is useful
 - can be discussed with stakeholders
 - used to eval design choices
 - good source of test cases
 - note: most obvious problem might not be right one to solve
- still need to check:
 - soln correctly solves the problem (**verification**)
 - problem stmt corresponds to stakeholder need (**validation**)



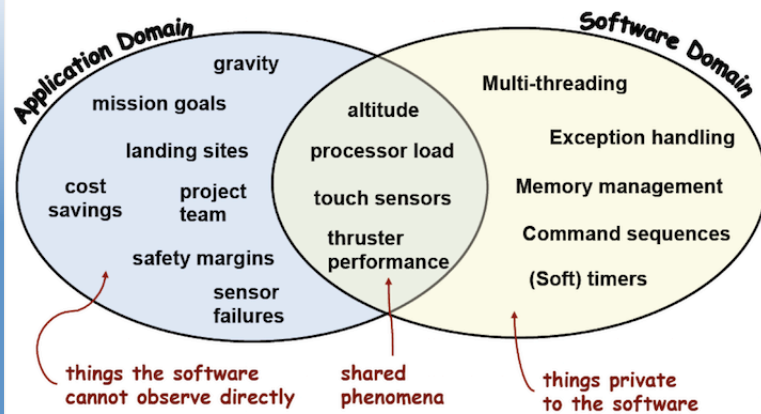
but, design changes the world...



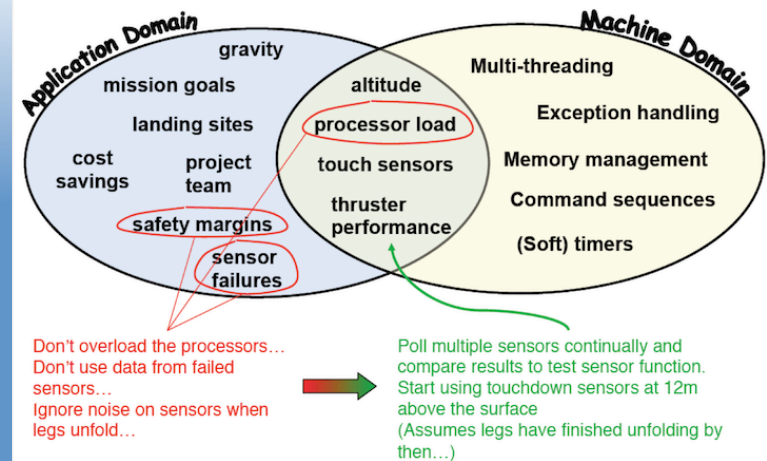
requirements as theories



example – landing on mars



example – landing on mars (2)



thinking about software requirements



- domain properties (assumptions):
 - things in domain that are true regardless if system is ever built
- (system) requirements:
 - things in the application domain we wish to be made true by building proposed system
 - may involve things which the machine can't access
- a (software) specification:
 - a desc of behaviours that the program must have to meet the requirements
 - can only be written in terms of the shared phenomena

fitness for purpose?

- two correctness (**verification**) criteria:
 - the software on a particular computer satisfies the specification
 - the specification, in context of domain properties, satisfies the requirements
- two appropriateness (**validation**) criteria:
 - enumerated all the appropriate requirements
 - properly characterized the relevant domain properties
- example:
 - requirement R: “reverse thrust shall only be enabled when the aircraft is moving on the runway”
 - domain properties D:
 - wheel pulses on \Leftrightarrow wheels turning
 - wheels turning \Leftrightarrow moving on the runway
 - specification S: “reverse thrust enabled \Leftrightarrow wheel pulses on”
 - verification: $S, D \Rightarrow R$



another example

- requirement R: “ the database shall only be accessible by authorized personnel”
- domain properties D:
 - authorized personnel have passwords
 - passwords are never shared with non-authorized personnel
- specification S: “access to the database shall only be granted after the user types an authorized password”

S, D \Rightarrow R

- but what if domain assumptions are wrong?

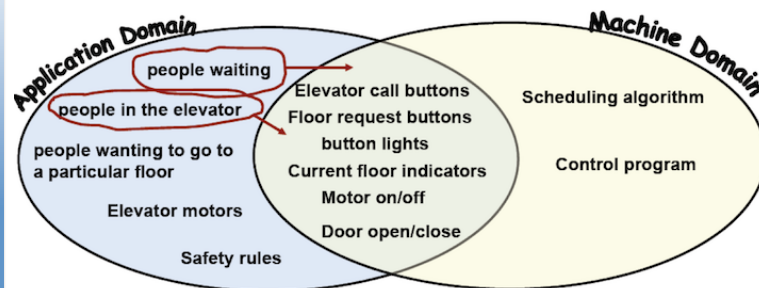
another example (2)

- people share passwords
- how to fix?
 - application domain only
 - user education – don’t do it!



move the boundaries

- ex. elevator control system:



- can shift things around:
 - add sensors to detect if people are waiting
 - changes the nature of the problem being solved

observations

- analysis is not necessarily a sequential process:
 - don’t have to have problem statement before soln statement
 - re-writing problem statement can be useful at any stage of development
 - but beware of the endlessly moving target
- the problem statement may (will) be imperfect
 - models are approximations of the world
 - will contain inconsistencies, will be missing info., assess the risk that these will cause serious problems

observations (2)

- perfecting a specification may not be cost effective
 - requirements analysis has a cost
 - for different projects the cost-benefit balance will be different
 - depends on consequences of getting it wrong
- problem statement should never be treated as fixed
 - change will happen, and must be planned for
 - should have a mechanism for incorporating changes periodically

stakeholders

- stakeholder analysis:
 - identify all people who must be consulted during info acquisition
- examples:
 - users: features and functionality
 - customers: best value for money
 - biz analysts / marketing team: “are we beating the competition?”
 - support staff: make it easy to use, learn & manage
 - tech writer: need to prepare manuals
 - project manager: on time, within budget, all requirements met

requirements to design

- requirements analysis:
 - It’s all about (correctly) identifying the purpose

what problem are we trying to solve?

- answer this wrong and you’ll have a quality fail (and all it’s associated nastiness)

starting point

- given a vague request for a new feature from users of your software:
 - identify the problem (**stakeholders, domain model**)
 - what is the goal/vision of those pushing for it?
 - scope the problem
 - how much of the vision do we need to tackle?
 - what is actually needed?
 - identify solution scenarios
 - (**use cases**) how will users interact with the software to solve the problem?
 - map onto the architecture (**robustness analysis**)
 - how will the needed functionality be met, what modules/classes will we need, code reuse?

what requirements analysts do

- given a “problem”...
 - some notion of a problem that needs solving
 - dissatisfaction with current system, new business opportunity, savings of: cost, time, etc.
 - requirements analyst is an agent of change
- ...the requirements analyst must:
 - identify the problem (or opportunity)
 - which problem needs to be solved? (**boundaries**)
 - where is the problem? (understand context/**domain**)
 - whose problem is it? (identify all **stakeholders**)
 - why does it need solving? (stakeholder **goals**)
 - when does it need to be solved? (identify development **constraints**)
 - what might prevent the solution? (**feasibility** and **risk**)
 - how might a software system help (collect **use cases**)

refresher



- domain properties (assumptions):
 - things in domain that are true regardless if system is ever built
- (system) requirements:
 - things in the application domain we wish to be made true by building proposed system
 - may involve things which the machine can't access
- a (software) specification:
 - a desc of behaviours that the program must have to meet the requirements
 - can only be written in terms of the shared phenomena

identifying actors

- ask the following questions:
 - who is primary user (**actor**) of the system?
 - who will need support for daily tasks
 - who/what has interest in results that the system produces?
 - who maintains & keeps system working? (secondary actor)
 - what hardware is required? with what other systems does it interact/depend?
- look for:
 - users who directly use the system
 - others that need services of the system

example – talk-while-u-type

```

tv.addTextChangedListener(new TextWatcher() {
    /** Characters that define a word boundary. */
    private static final String WORD_BOUNDARY = " .?!,:;)}\n\t";

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        if (((CheckBox) findViewById(R.id.kb_speak_while_typing)).isChecked()) {
            // Check that a single new char was appended, that's all I care about
            if (s != null && s.length() > 0 && before == 0 && count == 1 && start == s.length()-1) {
                if (WORD_BOUNDARY.indexOf(s.charAt(s.length() - 1)) != -1) { // finished a word, speak it!
                    String word = s.toString().substring(0, s.length()-1);

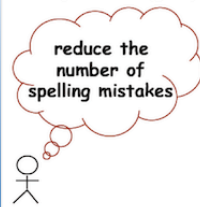
                    // find start of the word
                    int startOfWord = -1;
                    for (int i = word.length()-1; i >= 0; i--) {
                        if (WORD_BOUNDARY.indexOf(word.charAt(i)) != -1) {
                            startOfWord = i+1; break;
                        }
                    }
                    if (startOfWord == -1) startOfWord = 0; // first word

                    // isolate the word to speak
                    word = word.substring(startOfWord, word.length());
                    if (word.length() > 0) MyVoiceApp.speak(word);
                }
            }
        }
    }
});
    @Override public void afterTextChanged(Editable s) {}
    @Override public void beforeTextChanged(CharSequence s, int start, int count, int after) {}
});

```

example – make less spelling mistakes

A requirement (goal)



Functions

- identify complete word
- highlight misspellings
- suggest words from dictionary
- pop up a menu

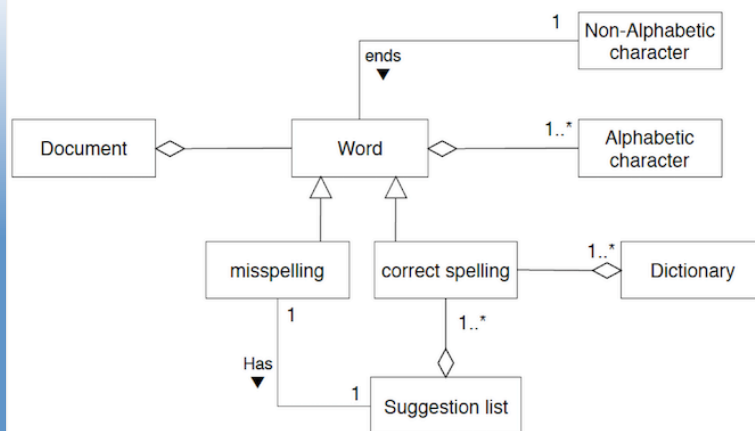
A Use Case

The user types text as usual. When the user completes each word, the system looks it up in the dictionary. If it is not in the dictionary, the word is underlined in red. The user can click on any underlined word, to see a popup menu of suggested alternatives. Clicking any of these alternatives causes it to replace the original word.

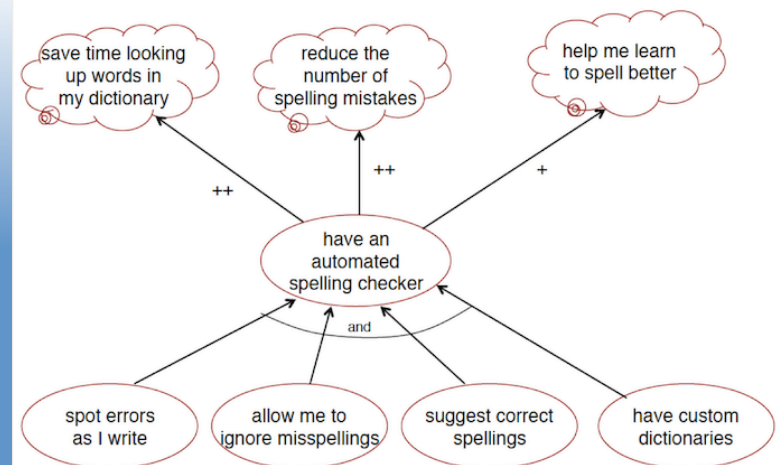
Domain Concepts



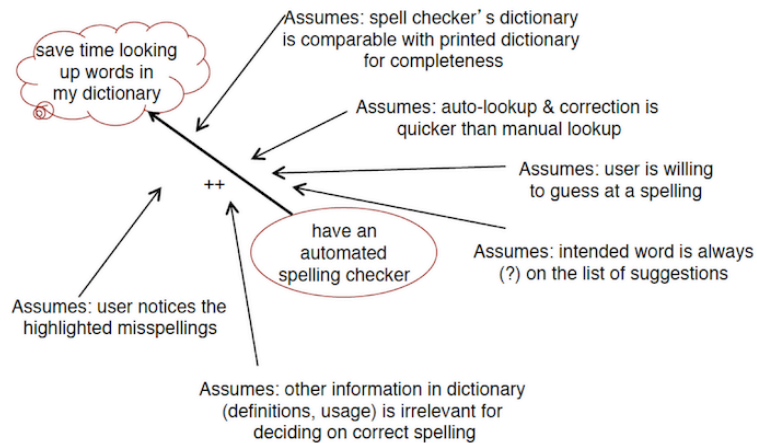
domain model



exploring goals



obstacle analysis



some requirements emerge

- functional requirements
 - user can see definitions for suggested spellings
 - user can add custom dictionaries
 - user can add new words to custom dictionary
 - user can tell spell checker to ignore some words
- quality requirements
 - dictionary should be comprehensive (as a printed one)
 - checking and suggesting should be fast
 - highlighted misspellings must be clearly visible

use case diagram

