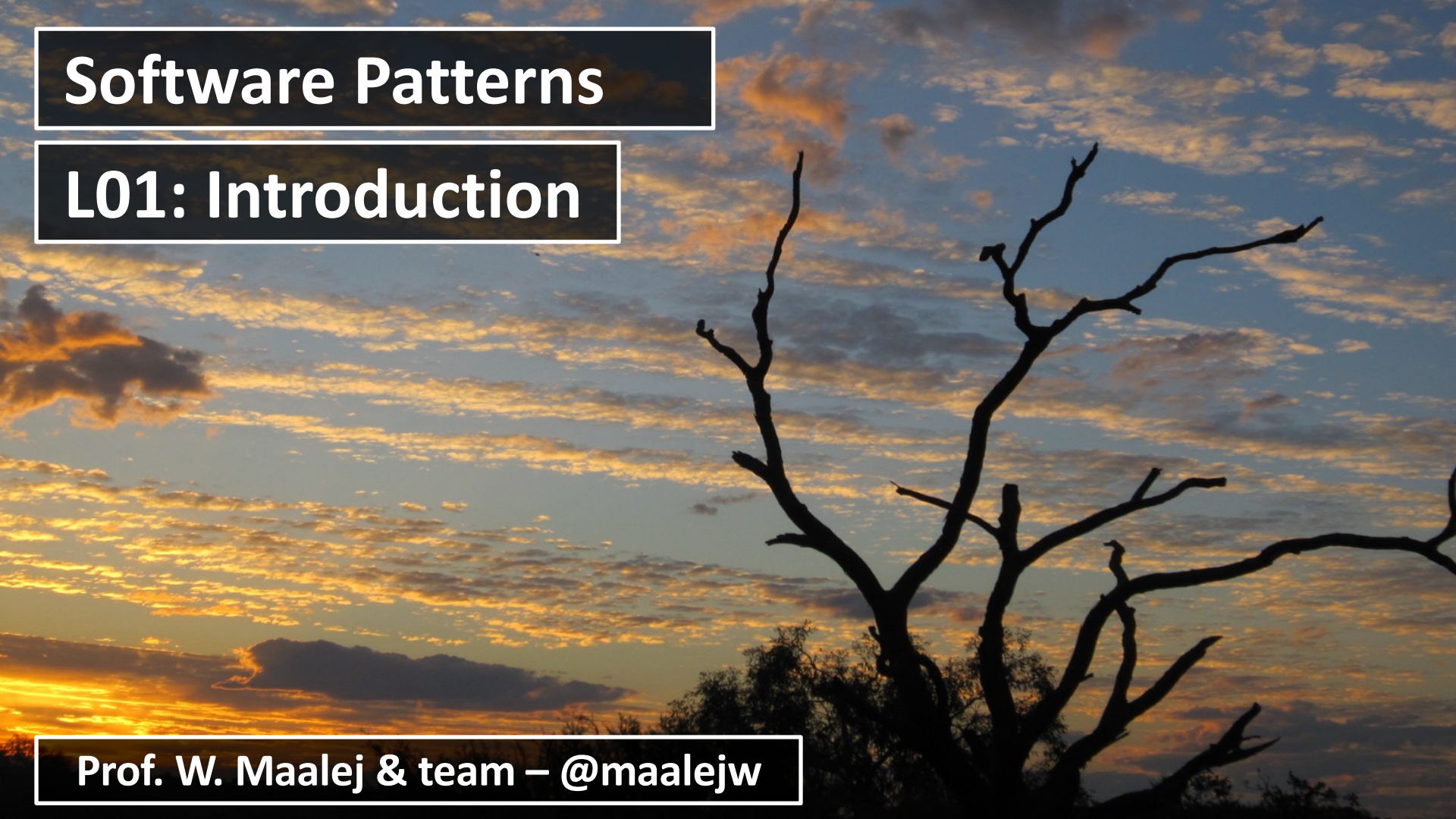
Software Patterns

L01: Introduction



Prof. W. Maalej & team – @maalejw

Overview

1. **Motivation and objectives**
2. Patterns: definition and vocabulary
3. SE Knowledge

What is this?

1.Nf3 d5 2.c4 c6 3.b3 Bf5 4.g3 Nf6 5.Bg2 Nbd7 6.Bb2 e6 7.O-O Bd6 8.d3 O-O 9.Nbd2 e5 10.cxd5 cxd5 11.Rc1 Qe7 12.Rc2 a5 13.a4 h6 14.Qa1 Rfe8 15.Rfc1

- A chess game!
- The language which is used here is called a *“domain-specific language”*

Another view

Lasker

Reti

This is a snapshot from the game Reti vs. Lasker, New York 1924

Comment from a chess master:

"*Lasker has the center, but Reti has **fianchettoed**, so he has the advantage….*"

Fianchetto is one of the basics of chess knowledge

Lasker

Reti

There is a pattern behind this!

Called the fianchetto: Usually taught in books about chess openings

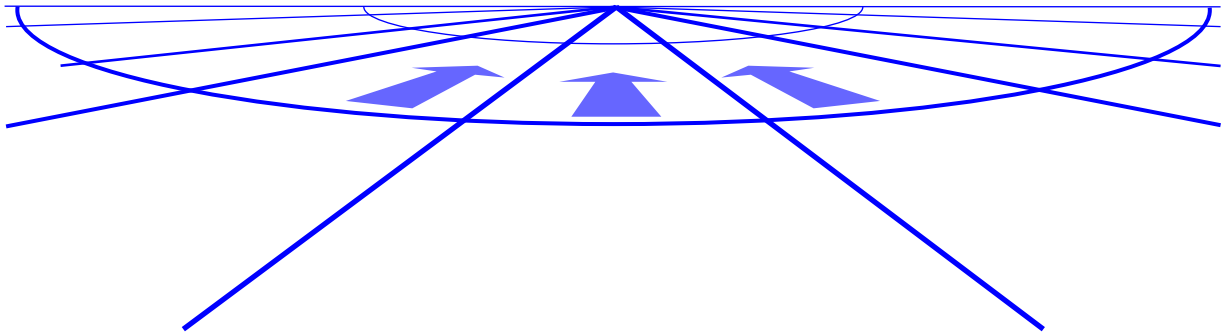
Comment in a typical newspaper chess column:
“We can see that Reti has allowed Lasker to occupy the centre but Reti has fianchettoed both Bishops to hit back at this, and has even backed up his Bb2 with a Queen on a1.”

Focus of this course
“patterns in software engineering”

- Patterns are a fundamental concept
- Helpful in managing **software engineering knowledge**
- They are useful “knowledge pieces”
- Help to build complex software systems in the context of frequent change by:
  - Reducing complexity and
  - Isolating the change

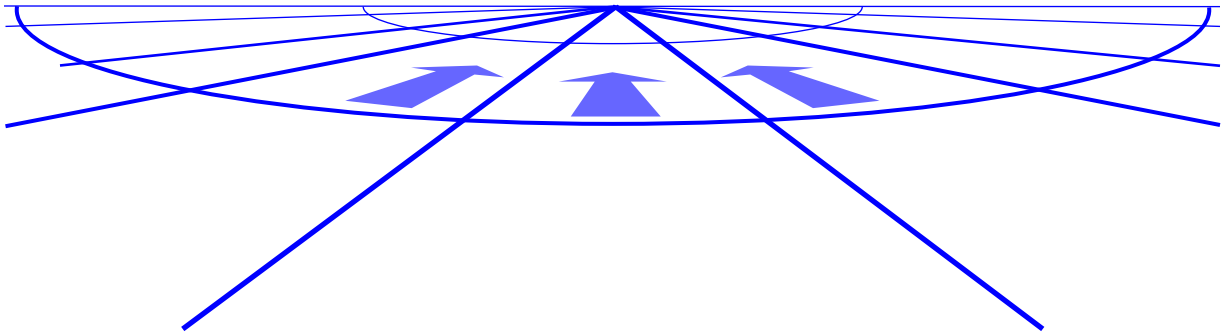
Objectives of this course

- Understand how to use **patterns** to develop high **quality** software within budgeted **resources** and while **change** occurs
- Acquire **technical knowledge** (patterns in development activities)
- Acquire **management knowledge** (patterns in management activities)
- Apply empirical methods to software data for **deriving and evaluating** (your own) software patterns



Objectives of the module "Empirical Software Engineering"

1. Basic knowledge in **empirical methods** and how they can be **applied** in the field of software engineering
2. Applying empirical research to manage **complex systems** (data driven decisions)
3. Insights about the state of the art and **advance research topics** in software engineering and application



What is empirical research?

+

Observation **Data**

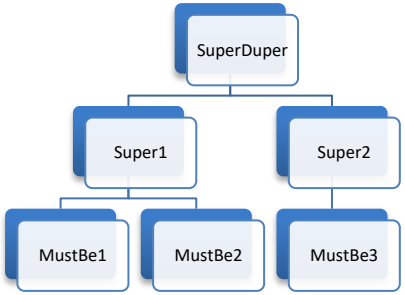
Systematic

The “new standard” in the SE community!

Other research approaches

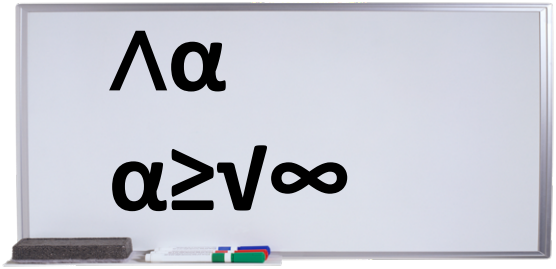
Engineering-driven

Analytical

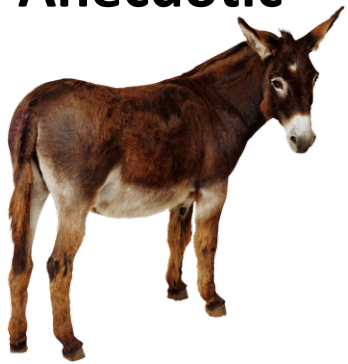


...

Mathematical, formal



Anecdotic



Goals of empirical studies

Explore

Understand phenomena and identify problems

Evaluate

Check/improve hypotheses, measure impact

Research strategies

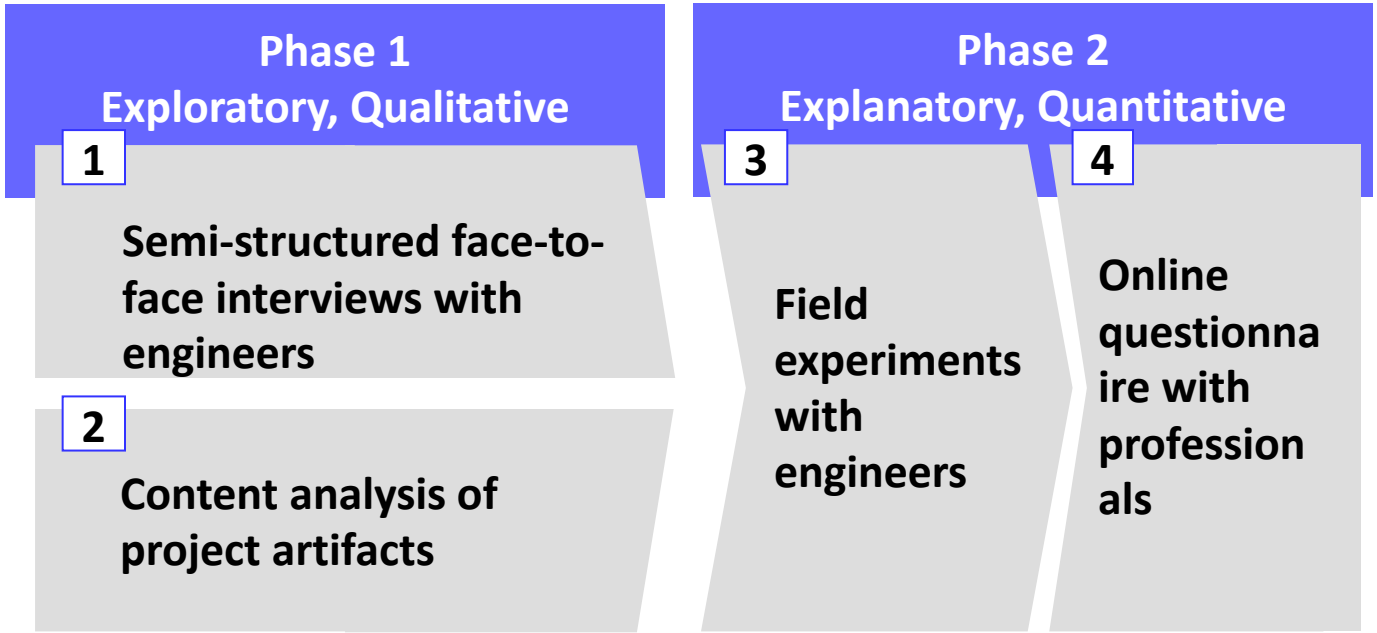
Qualitative

Quantitative

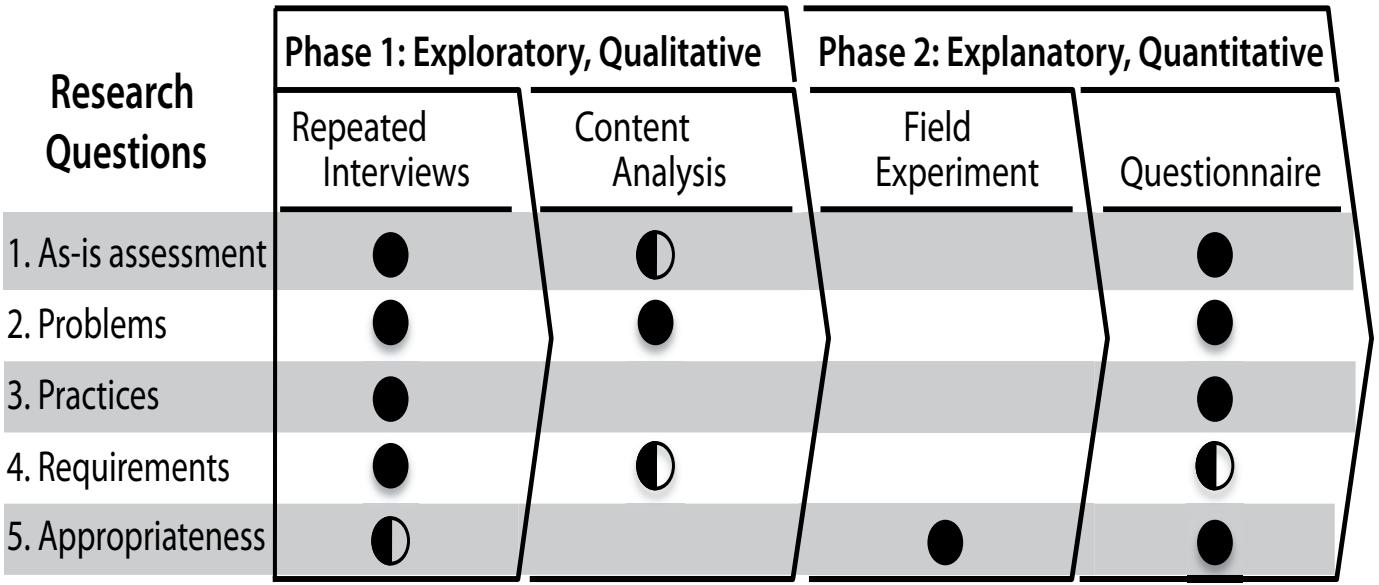
Both are important!

Combine: methodological pluralism, triangulation

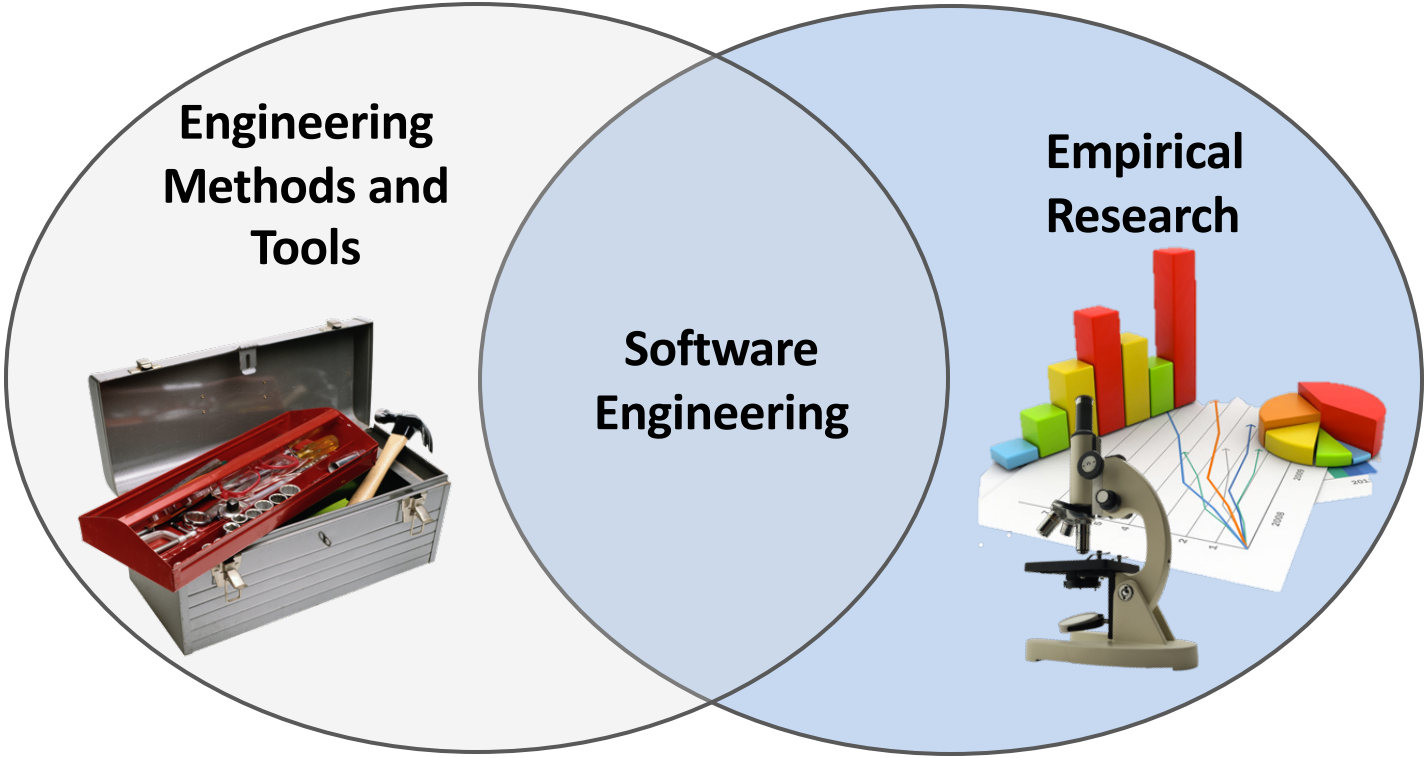
Example: Tool integration revisited [Maalej, 2009]



How it was presented in the paper



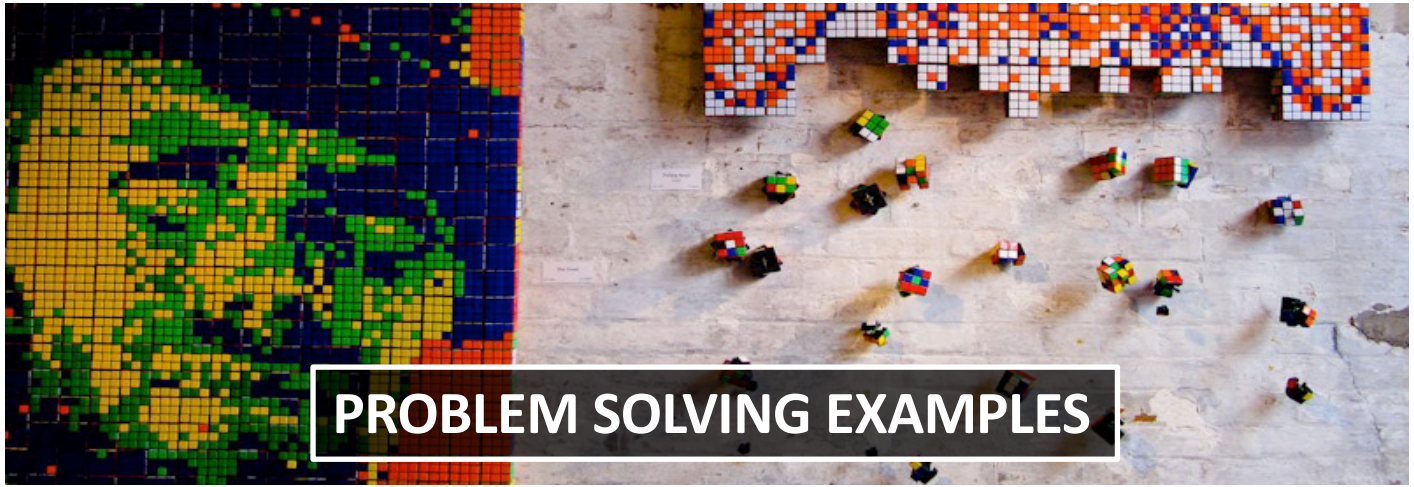
Software engineering & empirical research?



Software engineering is more than coding

- It is a **problem solving activity**
  - Understanding a problem
  - Proposing a solution and plan
  - Engineering a system based on the proposed solution using a ***good*** design

16



PROBLEM SOLVING EXAMPLES

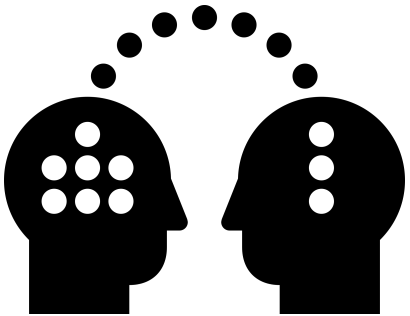
Software engineering is more than coding

- It is a **problem solving activity**
  - Understanding a problem
  - Proposing a solution and plan
  - Engineering a system based on the proposed solution using a ***good*** design
- It is about **dealing with complexity**
  - Creating abstractions and models
  - Distributing the tasks among different people
- It is about dealing with **change**
  - Requirements, design, implementation, testing, delivery, and maintenance

18

Software engineering is also…

- Knowledge management activity
  - Elicitation, analysis, design (creation), and validation of system- and process- knowledge
- Rationale management activity
  - Making decisions explicit to all stakeholders



Initial course schedule

Lecture	Main subject	Topics
L01	**Introduction**	Theory behind patterns
L02	**Foundation of OO Software Engineering**	Information hiding, Coupling/Cohesion, Polymorphism…
L03	**Structural Design Patterns**	Adapter, Bridge, Proxy, Composite
L04	**Behavioral and Creational Design Patterns**	Observer, Strategy, Template, (Abstract) Factory, Command
L05	**Architecture Patterns I**	MVC, Black board, Client-Server/ pep
L06	**Architecture Patterns II**	Client-Dispatcher-Server, Broker
L07	**Testing Patterns**	Unit Testing, Dependency Injection, Mocks, Mutation Testing
L08	**Quality Patterns**	*DD, CI, CD, Quality Assurance
L09	**Usability Patterns I**	Usability for Mobile: Patterns and Anti-Patterns
L10	**Usability Patterns II**	Usability for Mobile: Anti-Patterns
L11	**Collaboration and Management Patterns**	Work Allocation Patterns, Management Anti-Patterns
L12	**Knowledge and Documentation Patterns**	Taxonomy of API documentation

Rules of the game…

- This **Software Patterns** lecture
  - Moodle videos to watch by Wednesday every week
  - Wednesday Q&A session with lecturer + bonus questions
- Lecture **Software Requirements**
  - Moodle videos to watch by Wednesday every week
  - Monday Q&A session with lecturer + bonus questions
- Seminar "**Empirical Software Engineering**"
  - Paper reviewing, presentation and discussion
  - One presentation, at least one discussion per student
  - Multiple seminar groups, we will balance groups

Rules of the game…

- One written exam at the end of the semester
  - One part about requirements, one about patterns, one about empirical methods
- The admission requires the **active participation** in the seminar
- Material (including slides) in moodle https://lernen.min.uni-hamburg.de/ (**key: 'EMSE-21'**)
  - **EMSE 2021 – Seminar**
  - **EMSE 2021 – Lecture SP**
  - **EMSE 2021 – Lecture SR**
- Links and info on our website: mast.informatik.uni-hamburg.de/teaching/

Assumptions for this course

- **Assumptions:**
  - You can use an ***object-oriented*** programming language
  - You are familiar with a ***modeling language*** (UML)
  - You have taken a bachelor course
    "Introduction into software engineering" (or a similar)
- **Beneficial:**
  - You have experience in analysis & design
  - You have visited the lecture Software Architecture
  - You have experience with a large software system
  - You have participated in a large software project (e.g. M-Lab)
  - You have experience with quantitative methods or mining techniques

Overview

1 Motivation and objectives

2 **Patterns: definition and vocabulary**

3 SE Knowledge

How can we describe software engineering knowledge?

- Software engineering knowledge is not only a set of ***algorithms***
- It also contains a catalog of ***patterns*** describing generic solutions for recurring problems
  - Not describable in a programming language
  - Description usually in **natural language**
  - A pattern is presented in form of a schema consisting of sections of text and pictures (Drawings, UML diagrams…)

Algorithm vs. Pattern

Algorithm

- A method for solving a problem using a **finite sequence** of well-defined **instructions** for solving a problem
- Starting from an initial state, the algorithm proceeds through a series of successive states, eventually terminating in a final state

Pattern

- „A pattern describes a **problem** which occurs over and over again in our environment, and then describes the **core of the solution** to that problem in such a way that you can use this solution a million times over, without ever doing it the same way twice“

  [Alexander, A Pattern language]

Pattern definition

Original definition (C. Alexander):

A pattern is a three-part **rule**, which expresses a relation between a certain **context**, a **problem**, and a **solution**

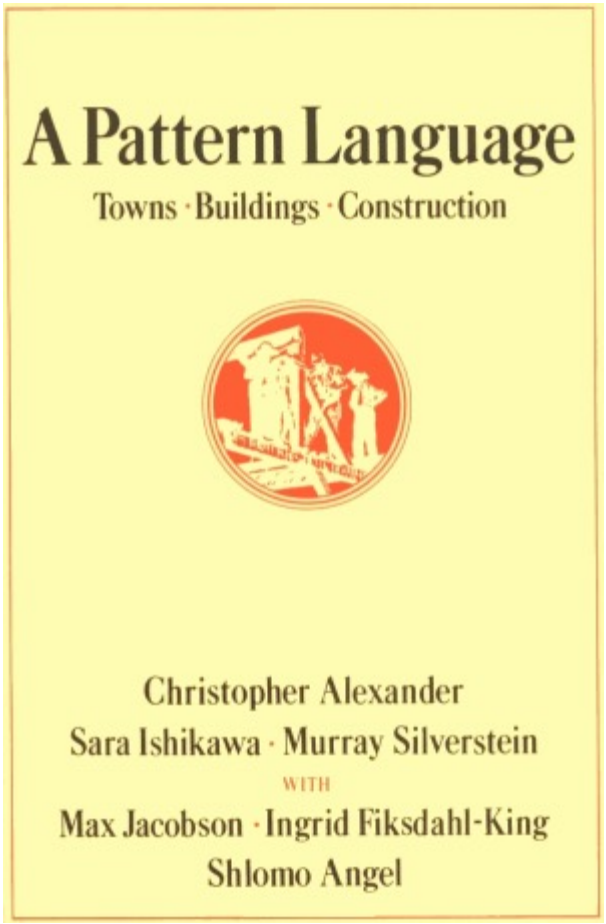
Refined Definition (R. Gabriel): A three-part rule, which expresses

- a relation between a certain context, and
- a certain system of forces which occurs repeatedly in that context, and
- a certain **software configuration** which allows these forces to resolve themselves.

[Gabriel, Patterns of Software: Tales From the Software Community]

Christopher Alexander

- 1936 Vienna, Austria
- More 200 building projects
- Creator of the „Pattern language“
- Professor emeritus at UCB



Patterns originated from architecture

- **C. Alexander's Philosophy:**
  - Buildings have been built for thousands of years by **users** who were **not architects**
  - Users **know more about what they need** from buildings and towns than architects
  - Good buildings are based on a set of **design principles** that can be described with a pattern language

Winchester house

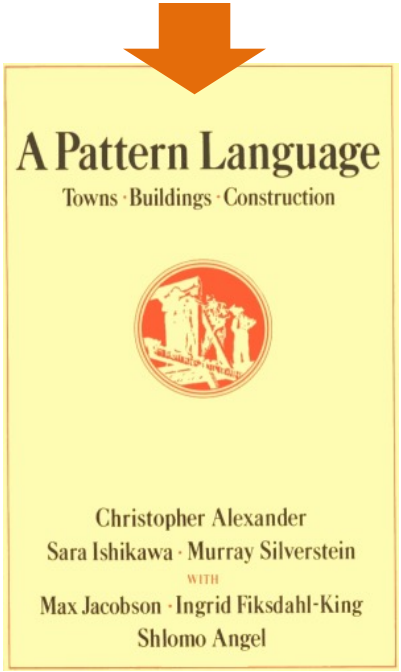
- In San Jose
- 160 rooms, including 40 bedrooms and 2 ballrooms, 47 fireplaces, 10,000 window panes, 17 chimneys, 2 basements and 3 elevators
- **No architect was ever consulted**
- Continuously built for 38 years (1884-1922)
- Originally 7 floors, now 4

http://www.winchestermysteryhouse.com/

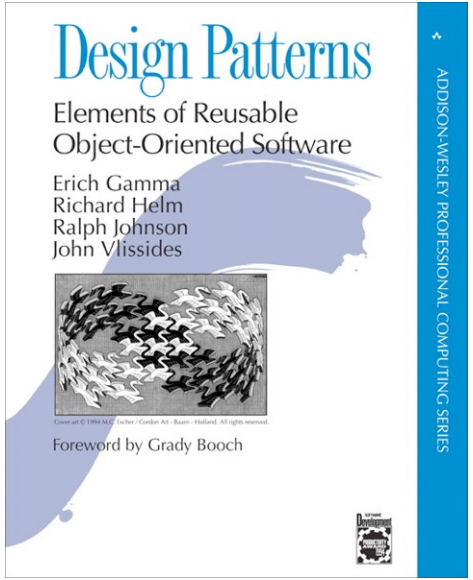
However: end users need architects…

…Desperately

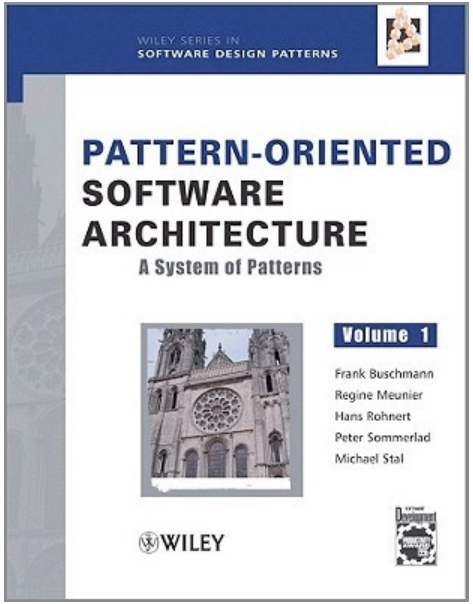
Schemata for describing patterns



A Pattern Language – Towns Buildings Construction, Christopher Alexander, Sara Ishikawa, Murray Silverstein, Oxford University Press, 1977



Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Addison Wesley, 1994



Pattern-Oriented Software Architecture - A System of Patterns, Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Wiley, 1996

Alexander's schema ("Alexandrian form")

- Parts (Sections)
  - Name of the Pattern
  - Picture of an example for the pattern
  - Context
  - Problem: Short description and elaborate description
  - Solution: Description and Diagram
  - Conclusion
- Sections without explicit headings and are separated by symbols
  - 3 diamonds between context/problem and after the solution
  - 3 diamonds between solution and conclusion
- Keyword "**therefore**" to separate the problem from the solution

Example of a pattern (Alexander's schema)

- Pattern Language on-line
  - Towns
  - Buildings
  - Construction
- A Child's History of Pattern Language
- Language builder
- Buy a copy of the book

home page >> patterns

- The overall arrangement of a group of buildings
- The position of individual buildings
- Entrances, gardens, courtyards, roofs and terraces
- Paths and squares
- Gradients and connection of space
- The most important areas and rooms (in offices, workshops and public buildings)
- Knit the inside of the building to the outside
- Arrange the gardens, and the places in the gardens
- Inside, attach necessary minor rooms and alcoves
- Fine tune the shape and size of rooms and alcoves
- Give the walls some depth

http://www.patternlanguage.com/leveltwo/patterns.htm

Patterns

The most important areas and rooms (in offices, workshops and public buildings)

Outbuildings and access to the street and gardens

146 FLEXIBLE OFFICE SPACE

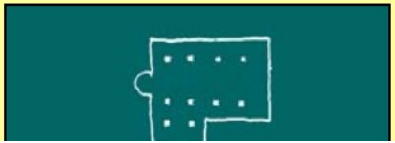
. . . imagine that you have laid out the basic areas of a workshop or office SELF GOVERNING WORKSHOPS AND OFFICES (80), OFFICE CONNECTIONS (82). Once again, as in a house, the most basic layout of all is given by INTIMACY GRADIENT (127) and COMMON AREAS AT THE HEART (129). Within their general framework, this pattern helps to define the working space in more detail, and so completes these larger patterns.

Is it possible to create a kind of space which is specifically tuned to the needs of people working, and yet capable of an infinite number of various arrangements and combinations within it?

Therefore:

Lay out the office space as wings of open space, with free standing columns around their edges, so they define half-private and common spaces opening into one another. Set down enough columns so that people can fill them in over the years, in many different ways - but always in a semipermanent fashion.

If you happen to know the working group before you build the space, then make it more like a house, more closely tailored to their needs. In either case, create a variety of space throughout the office - comparable in variety to the different sizes and kinds of space in a large old house.



151 SMALL MEETING ROOMS

Name of the Pattern

. . . within organizations and workplaces - UNIVERSITY AS A MARKETPLACE (43), LOCAL TOWN HALL (44), MASTER AND APPRENTICES (83), FLEXIBLE OFFICE SPACE (146), SMALL WORK GROUPS (148), there will, inevitably, be meeting rooms, group rooms, classrooms, of one kind or another. Investigation of meeting rooms shows that the best distribution - both by size and by position - is rather unexpected.

Problem

Picture of an example for the pattern

The larger meetings are, the less people get out of them. But institutions often put their money and attention into large meeting rooms and lecture halls.

Therefore:

Separation of Problem from Solution

Make at least 70 per cent of all meeting rooms really small - for 12 people or less. Locate them in the most public parts of the building, evenly scattered among the workplaces.

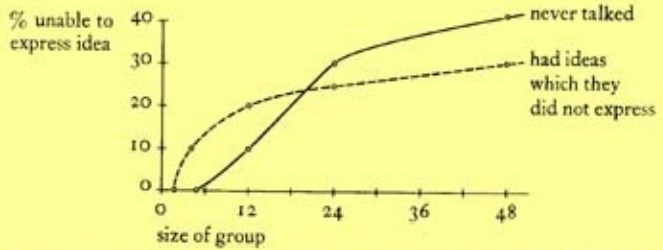
Solution: Description

Solution: Diagram

Context

Separation between Solution and Context

We first discuss the sheer size of meetings. It has been shown that the [illegible] the number who never talk, and the number who feel they have ideas which they have not been able to express. For example, Bernard Bass (*Organizational Psychology,* Boston: Allyn, 1965, p. 200) has conducted an experiment relating group size to participation. The results of this experiment are shown in the following graph.



As size of group grows, more and more people hold back.

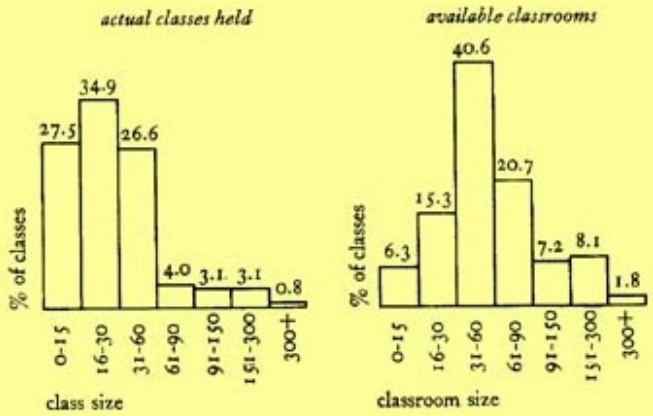
There is no particularly natural threshold for group size; but it is clear that the number who never talk climbs very rapidly. In a group of 12, one person never talks. In a group of 24, there are six people who never talk.

We get similar thresholds when we consider comfortable distances for talking. Edward Hall has established the upper range for full casual voice at about 8 feet; a person with 20/20 vision can see details of facial expression up to 12 feet; two people whose heads are 8 to 9 feet apart, can pass an object if they both stretch; clear vision (that is, macular vision) includes 12 degrees horizontally and 3 degrees vertically - which includes one face but not two, at distances up to about 10 feet. (See Edward Hall, *The Silent Language,* New York: Doubleday, 1966, pp. 118-19.)

Thus a small group discussion will function best if the members of the group are arranged in a rough circle, with a maximum diameter of about 8 feet. At this diameter, the circumference of the circle will be 25 feet. Since people require about 27 inches each for their seats, there can be no more than about 12 people round the circle.

The following histograms show the relative numbers of different sized classes held at the University of Oregon in the Fall of 1970 and the relative numbers of available classrooms in the different size ranges. We believe these figures are typical for many universities. But it is obvious at a glance that there are too many large classrooms and too few small classrooms. Most of the classes actually held are relatively small seminars and "section" meetings, while most of the classrooms are in the 30 to 150 size range. These large classrooms may have reflected the teaching methods of an earlier period, but apparently they do not conform to the actual practice of teaching in the 1970's.

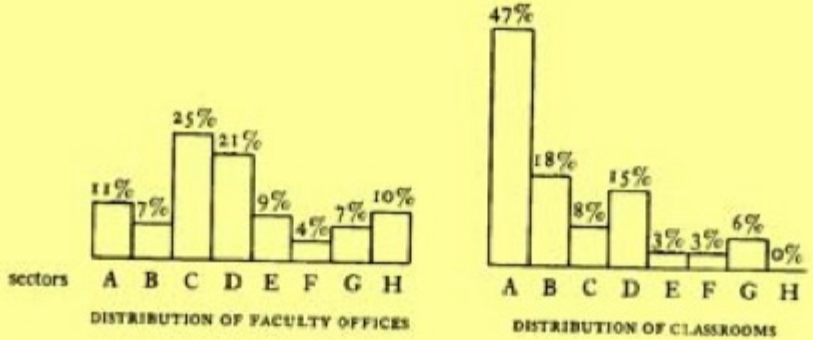
More Context



Histogram: Classes don't fit the classrooms.

We found that the meetings of official committees, boards, and commissions in the City of Berkeley have a similar distribution. Among the various city boards, commissions, and committees, 73 per cent have an average attendance of 15 or less. Yet of course, most of these meetings are held in rooms designed for far more than 15 people. Here again, most of the meetings are held in rooms that are too large; the rooms are half-empty; people tend to sit at the back; speakers face rows of empty seats. The intimate and intense atmosphere typical of a good small meeting cannot be achieved under these circumstances.

Finally, the *spatial* distribution of meeting rooms is often as poorly adapted to the actual meetings as the size distribution. The following histograms compare the distribution of classrooms in different sectors of the University of Oregon with the distribution of faculty and student offices.



The meeting rooms are not located where people work.

Once again, this discrepancy has a bad effect on the social life of small meetings. The meetings work best when the meeting rooms are fairly near the participants' offices. Then discussions which begin in the meeting rooms are able to continue in the office or laboratory. When the meeting rooms are a long walk from offices, the chances of this kind of informal business are drastically reduced.

Conclusion 151 small meeting rooms

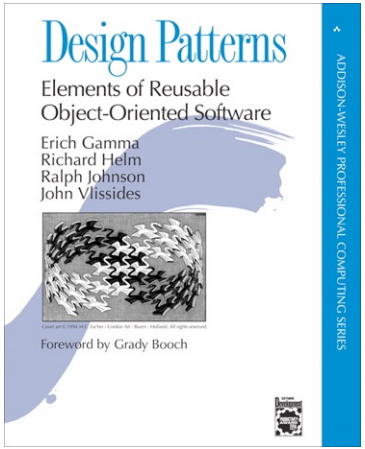
Shape meeting rooms like any other rooms, perhaps with special emphasis on the fact that there must be no glare - LIGHT ON TWO SIDES OF EVERY ROOM (159) - and on the fact that the rooms should be roughly round or square, and not too long or narrow - SITTING CIRCLE (185). People will feel best if many of the chairs are different, to suit different temperaments and moods and shapes and sizes - DIFFERENT CHAIRS (251). A light over the table or over the center of the group will help tie people together - POOLS OF LIGHT (252). For the shape of the room in detail, start with THE SHAPE OF INDOOR SPACE (191). . . .

Gang of four schema:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

- Pattern Name and Classification
- Intent
- Also Known As
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample Code
- Known Uses
- Related Patterns

See: Pages 6-7
[Gamma et al 95]



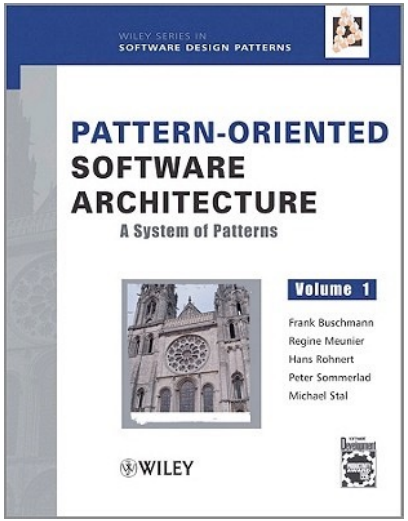
Gang of five schema:

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal

- Name
- Also Known As
- Example
- Context
- Problem
- Solution
- Structure
- Dynamics
- Consequences
- Implementation

- Sample Code
- Known Uses
- Related Patterns

See: Pages 20-21
[Buschmann et al 96]



Pattern language and pattern catalogs

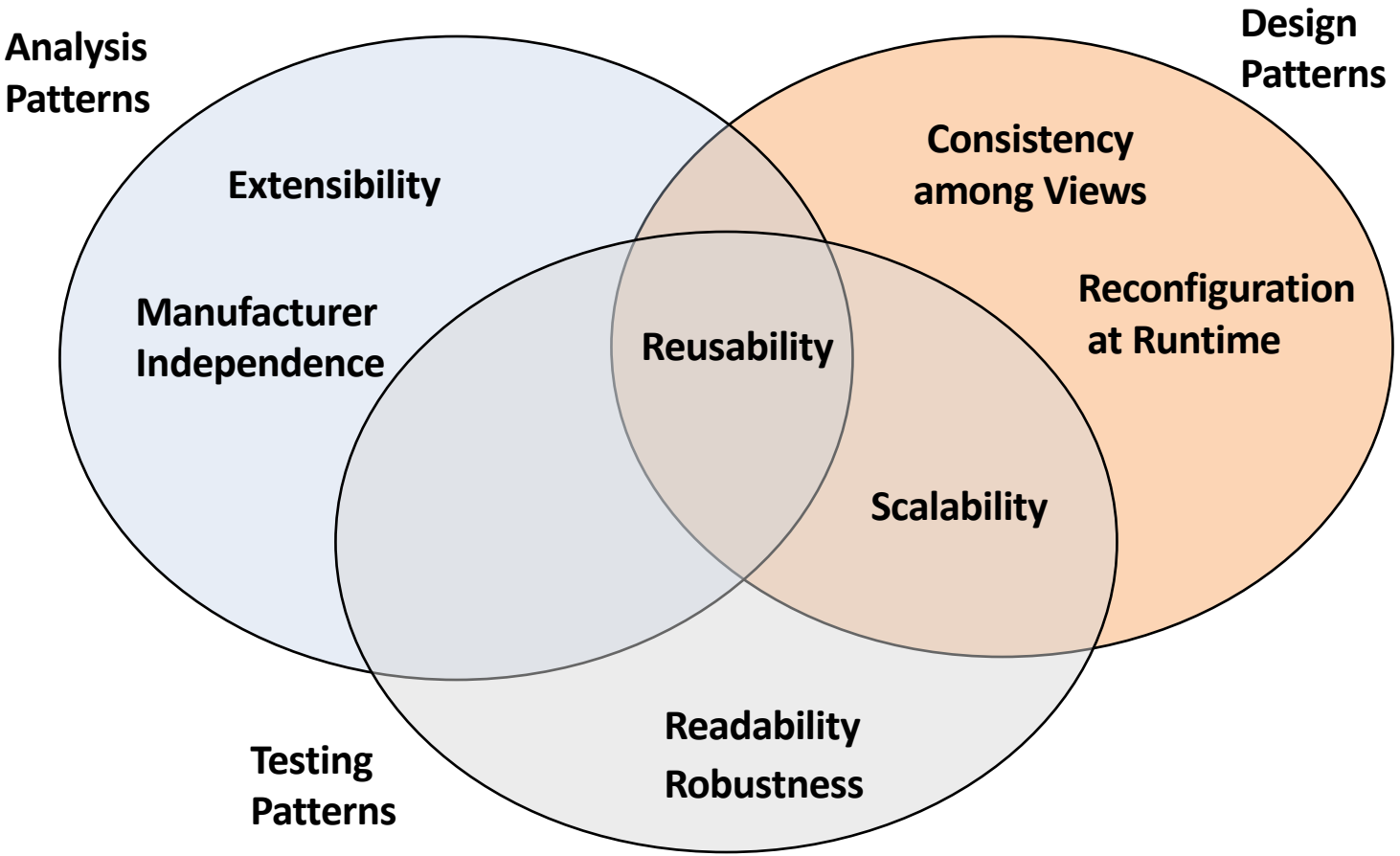
Pattern Language:

- A **vocabulary** for understanding and communicating ideas behind patterns
- The rules to **combine** a collection of patterns into an “architectural style”

Pattern Catalog:

- A **collection** of related patterns
- It typically subdivides the patterns into a small number of broad **categories**
- May include cross **referencing** between patterns

Patterns often address nonfunctional requirements



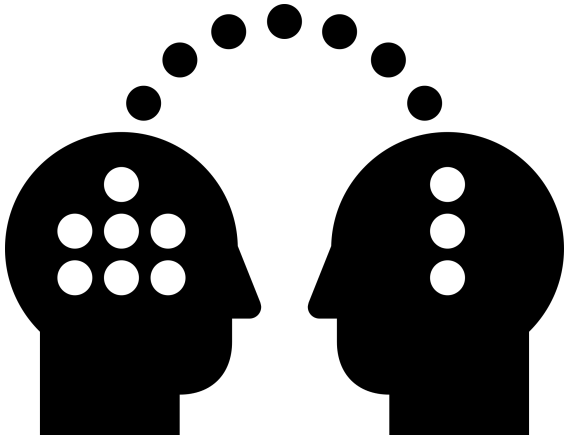
Overview

1 Motivation and objectives

2 Patterns: definition and vocabulary

3 **SE Knowledge**

The fundamental question for this class:

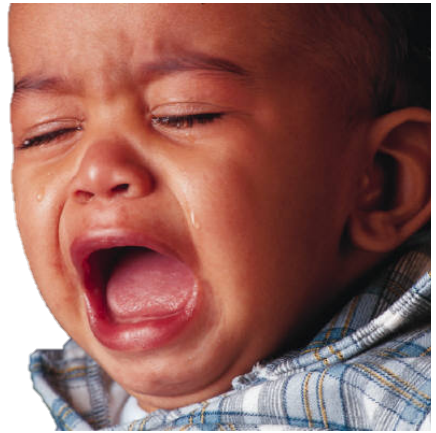


How do we acquire and describe software system, software project, and software engineering knowledge?

1. Insight: Knowledge is falsifiable

- Karl Popper ("Objective Knowledge):
  - There is **no absolute truth** when trying to understand reality
  - One can only build theories, that are "true" until somebody finds a counter example
- **Falsification:** The act of disproving a theory or hypothesis
- The truth of a theory is never certain
- We must use phrases like:
  - "based this data", "by our best judgment", "using state-of-the-art knowledge"

Consequence: software engineering knowledge is also falsifiable



- In software engineering any model is a theory
- We build models and try to find counter examples by:
  - Requirements validation, user interface testing, design review, unit/system testing…

Testing: The act of disproving a model

Evaluation: Testing hypothesis by using empirical data

2. Insight: Failures are helpful

„Success in Engineering
is defined by its failures“

Petrovski

Petrovski’s paradoxical approach to design:

- Better information comes from designs that fail than from those that succeed.
- Reason: failures draw more scrutiny (critical observation or examination).
- Petrovski says without failure, complacency sets in.

Petrovski:

- Failures in civil engineering helped to develop the discipline

- Early difficulties were experienced with iron bridges in Victorian times
- Each failure led to a period of over-engineering stronger bridges with a greater margin of error

3. Insight: Science can help to create knowledge

- Generally: empirical methods
- Grounded Theory is the **systematic** methodology that involves the **generation of theory** (knowledge) **from data**
- Qualitative research method
- Can be used to identify patterns

*"Most grounded theorists believe they are theorizing about **how the world 'is'** rather than **how people see it**" [Steve Borogatti]*

Summary

1. Patterns are knowledge: Reusable ideas for solving recurrent problems in certain contexts

2. We acquire, extract, and describe knowledge to solve recurring problems

3. Knowledge is acquired through failure (Learning from failures) or through empirical research (observation& data)

4. This course focuses on software patterns, design, testing, usability, and management patterns