COMP 1531
Software Engineering Fundamentals

Course Introduction

Aarthi Natarajan

COMP 1521 17s2
Software Engineering Fundamentals
Our Team
LiC: Aarthi Natarajan
a.natarajan@unsw.edu.au

Web: http://webcms3.cse.unsw.edu.au/COMP1531/17s2/

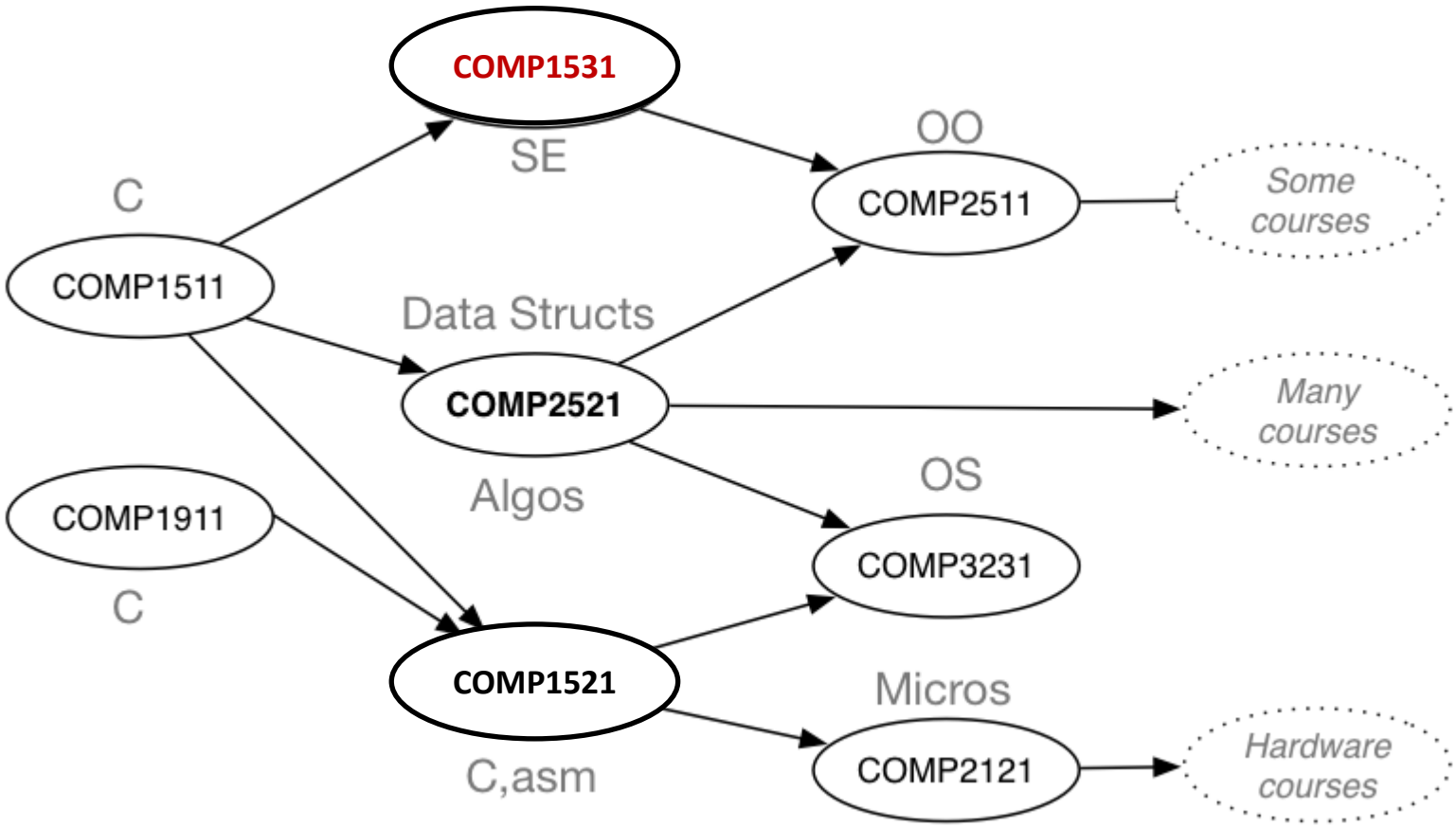
Course Admin:
Deepanjan Chakrbarthy, Sajid Anower (GitHub)

Tutors & Lab Assistants:
Deepanjan Chakrbarthy, Sajid Anower, Jessica Theodosius, Matthew Phillips, Matthew Perry, Hussein Debel, Isaac Carr, Bella Mangunsong, Anna Azzam, Kongzhang Hao, Minjie Shen, Armin Chitizadeh, George Mountakis, Xiaocong Chen

COMP 1531 students

- Students in this course have completed **COMP 1511**
- Everyone has learned fundamental C programming and are familiar with variables, data types, loop structures, defining and using functions and returning results
- **COMP 1511**
  - Gets you thinking like a programmer
  - Solving problems by developing programs and expressing your solution in C
- **COMP 1531 Course Goals**
  - gets you thinking like a **software engineer**
  - teaches you how to deliver value to customers
  - achieve customer goals through solving problems by applying the fundamental principles of software engineering
  - **software engineering is NOT programming !!**
  - expose you to Python/Flask/Jinja2 framework

Course Context



COMP 1531 students

- Students in this course have completed **COMP 1511**
- Everyone has learned fundamental C programming and are familiar with variables, data types, loop structures, defining and using functions and returning results
- **COMP 1511**
  - Gets you thinking like a **programmer**
  - Solving problems by developing programs and expressing your solution in C
- **COMP 1531 Course Goals**
  - gets you thinking like a **software engineer**
  - teaches you how to deliver value to customers
  - achieve customer goals through solving problems by applying the fundamental principles of software engineering
  - **software engineering is NOT programming !!**
  - expose you to Python/Flask/Jinja2 framework

COMP 1531 Major Themes

- Understand the importance of Software Engineering and why Software Engineering is not programming
- Explore the key phases of software engineering life-cycle
- Analyse a problem and elicit user requirements
- Design using sound design principles
- Effective coding and testing techniques
- Team collaboration and software configuration management
- Understand and gain practical experience in Agile Software Development

Text Book

- There is no textbook.
- Material has been drawn from:
  - *Software Engineerings* , by Ivan Marsic, Rutgers, The State University of New Jersey (Available for free download from: http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf)
  - *Agile Software Development: Principles, Patterns and Practice* , by Robert C Martin, Pearson
  - *Foundations of Software Engineering,* by Ashfaque Ahmed and Bhanu Prasad, CRC Press
- Each week, the relevant chapters from the above sources will be highlighted
- Links to useful tutorials will be uploaded as necessary

System

- Most work done on Linux…
  - Lab work done on the CSE machine labs
  - Technology stack – Python, Flask, SQLite3 and SQLAlchemy (optional)
  - **Group project can be done on own equipment, but must use the above core technology stack**
  - Must use Python 3.5 onwards (Use virtual environment, more instructions will be provided in the forth-coming lectures and tutorial sessions)
  - Use your own favourite text editor (e.g., vim)
- Collaboration and Versioning Tool - GitHub
  - **You must create a GitHub account before Sunday this week, else you will not be able to do lab in week 02 !**

Classes

- Lectures…
  - 3 hours/week, weeks 1 – 12, Tue (14:00 – 16:00), Wed (15:00-16:00)
  - All lectures will be recorded (Echo 360)
- Tutorials…
  - 1 hour/week, weeks 2 – 13
  - Tutorials are compulsory to attend
- Labs…
  - 2 hours/week, weeks 2 – 13
  - small practical programming exercises, done in pairs (“pair programming”)
  - group project iteration demos scheduled in some lab sessions

Assessments - Group Project

- One main group project that runs through the whole term
  - Project specification will be released in Week 02
  - You are required to form groups of 4 (no more than 4)
  - Contributes to 25% of the final course mark
- Project is implemented using an ***Agile Software Development Model***
  - Working software to be delivered in iterations
  - Project specification **can change** at the end of an iteration (as customer changes mind…), so **design well** !!
  - Iteration demos will be held during your lab session, late demos will not be accepted
  - First iteration deadline will be advised next week
  - Marks will be awarded for each iteration demo, which will count towards your overall group project mark
  - Responsibilities to be assigned to each group member during each iteration
  - Final project demo held in week 13

Assessments

- Practical lab sessions (15% of overall mark)
  - Mostly done in pairs – “pair programming”, but occasional individual exercises (e.g., git-hub hands-on exercise)
  - Lab exercise for Week X **must** be done in the lab and demonstrated to tutor during Week X lab
  - Lab exercises will enable you to become familiar with key technology stack for COMP 1531 (and later..)
  - **Cannot** obtain marks by emailing solution to tutors
- Online Quizzes (5% towards final mark)
  - Multiple-choice format
  - Review of content covered in lectures
  - Taken in your own time (via WebCMS3)
  - Due before Sunday 11:59 at end of week
- Final Exam (55% towards final mark)
  - 3 hour final exam, more details through the term

Course Mark

- Course Work Mark
  - Quiz Mark + Lab Mark + Group Project Mark - (out of 45)
- Exam Mark
  - Mark from the 3 hour final exam (out of 55)
  - ExamOK = Exam Mark >= 24/55
- Final Course Mark (out of 100)
  - Course Work Mark + Exam Mark
- Final Grade
  - UF, If !ExamOK
  - FL, if Final Course Mark < 50/100
  - PS, if 50/100 ≤ Final Course Mark < 65/100
  - CR, if 65/100 ≤ Final Course Mark < 75/100
  - DN, if 75/100 ≤ Final Course Mark < 85/100
  - HD, if Final Course Mark ≥ 85/100

Supplementary Exam

❖ Students are eligible for a Supplementary Exam if and only if:

- they cannot attend the final exam due to illness or misadventure
- their final mark is in the range $47 \leq$ Final Course Mark < 50 (in this case, Final Course Mark is limited to 50)
- a supplementary exam will not be awarded for any other reason.

Tasks this week…

- GitHub
  - Create a GitHub account
  - After you have created your GitHub account, please go to this link: https://dry-savannah-14861.herokuapp.com/ to add yourselves to COMP 1531's GitHub organisation.
  - Put in your username. Note that your username would typically look something like 'username456'. If you put in your user email instead, the app app will silently fail and you won't receive an invitation to the GitHub organisation.
  - Then, go to https://github.com/orgs/cse1531S1/invitation and accept the invitation.
  - If you encounter any problem while doing this, either post it in the course forum (include your GitHub username and user email) or email **Sajid (s.anower@student.unsw.edu.au)** with your GitHub username and email.

More tasks this week…

- GitHub
  - Make yourself familiar with GitHub tutorials and lab02 that you will be performing “individually” in week 2
- Pair Programming
  - Start thinking about who your partner will be for “pair-programming” exercises
- Group Project
  - Start planning your team of **no more than 4**!
- Python
  - And check out the links for Python tutorials. This is a software engineering course, not a programming course !!

COMP 1531
Software Engineering Fundamentals

Introduction to Software Engineering

Aarthi Natarajan

What is software?

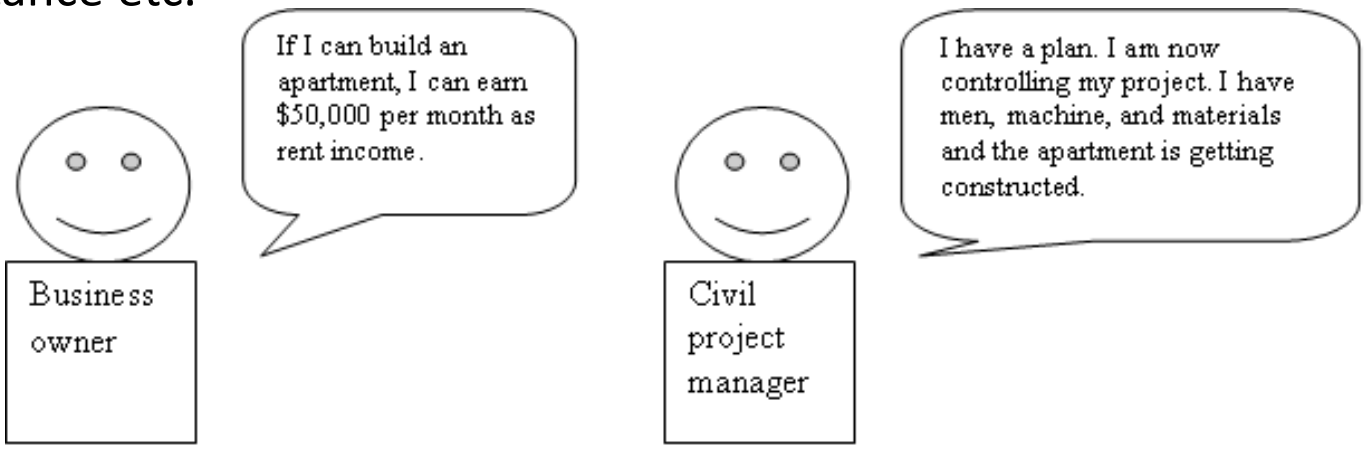
- A ***software*** is a program or sequence of instructions that tells the computer what tasks it needs to perform and how to perform them
- Software can be distinguished into:

  - ***application software*** such as a database, spreadsheet or word-processing program, a web browser, a console game etc.

  - ***system software*** that deals with operating the computer or devices connected to the computer (e.g., operating system such as Microsoft Windows, Mac OS, Unix or device drivers) or utility software (e.g., anti-virus programs)
- Software is everywhere and the economies of ALL developed nations are dependent on software
- A ***software product*** is often used to refer to a collective set of entities that includes software program, documentation, data…

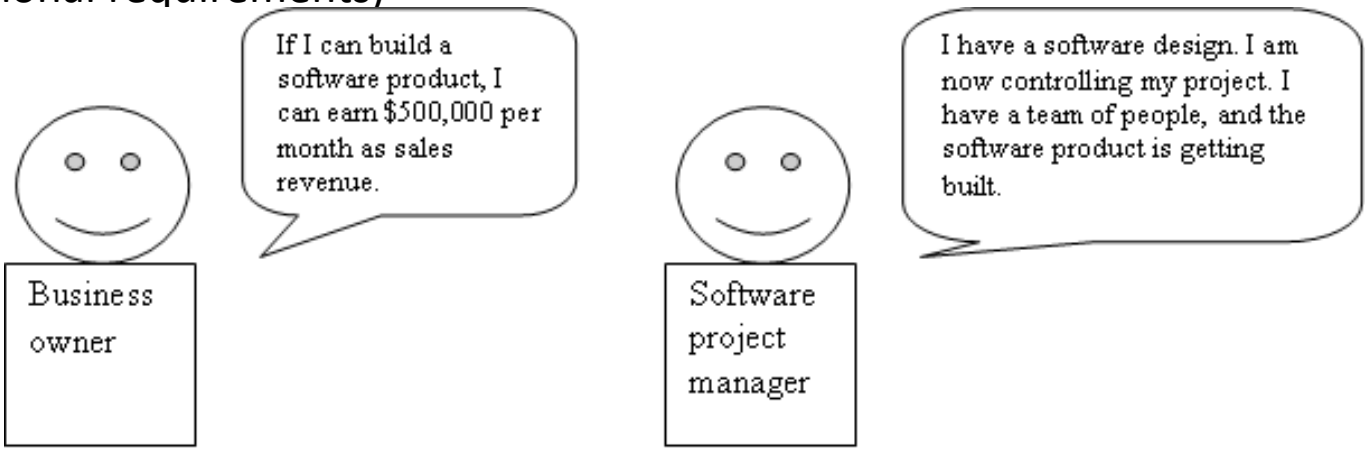
Civil Engineering Project

- **Objective:** A developer identifies a potential to make income (rental or sale)
- Developer initiates a project for building a complex of apartments and employs a team consisting of:
  - an architect to design the complex,
  - a civil project manager who estimates manpower, machinery and time required to finish the construction and makes a project plan, recruits people & machinery and oversees the project to ensure its completion as per the design and project plan.
  - Building needs to comply to standards such as fire-alarms, exits, earthquake resistance etc.



Software Development Project

- **Objective:** A business owner identifies the need for a software product because there is a market for the product or required for the enterprise itself
- A project is instituted for completion of the software product and a team is initiated consisting of:
  - Business analysts to outline the list of features of product
  - A software project manager who makes a project plan, estimates time, people (cost) to deliver the proposed product)
  - Software designers to design the product
  - Software developers to implement the project.
  - Software product tested and shipped.
  - Software product needs to comply with security, performance, reliability (non-functional requirements)



What is Software Engineering

IEEE's definition of Software Engineering:

" The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software. "

What is Software Engineering?

- "Software Engineering" is a discipline that enables customers achieve business goals through *designing* and *developing* software-based systems to solve their business problems e.g., develop a patient-record software in a doctor's surgery, a software to manage inventory
- A software engineer starts with a *problem definition* and applies tools of the trade to obtain a *problem solution*. However…
- Software engineering requires great emphasis on *methodology* or the *method* for managing the development process in addition to great skills with tools and techniques.

Software Engineering is not Programming

- Software engineering is often *confused* for programming
- Software engineering is about:
  - Understanding of the business problem definition
  - Creative formulation of ideas to solve the problem based on this understanding and designing the "blueprint" or architecture of the solution
- Programming is the craft of *implementing* the given "blueprint"
- A software engineer's focus is on *understanding* the interaction between the system-to-be and its users and the environment, and *designing* the software-to-be based on this understanding
- A programmer's focus is on the program code and ensuring that the code *faithfully* implements the design

Fundamental Law of Software Engineering

- Software engineer is willing to learn the problem domain (a problem cannot be solved without understanding it first). In particular, this task requires the ability to:
  - quickly learn new and diverse disciplines and business processes
  - communicate with domain experts, extract an abstract model of the problem from a stream of information provided in discipline-specific jargon, and formulate a solution that makes sense in the context of customer's business
  - design a software system that will realize the proposed solution and gracefully evolve with the evolving business needs for many years in the future

Role of a software engineer (1)

A bridge from customer needs to programming implementation

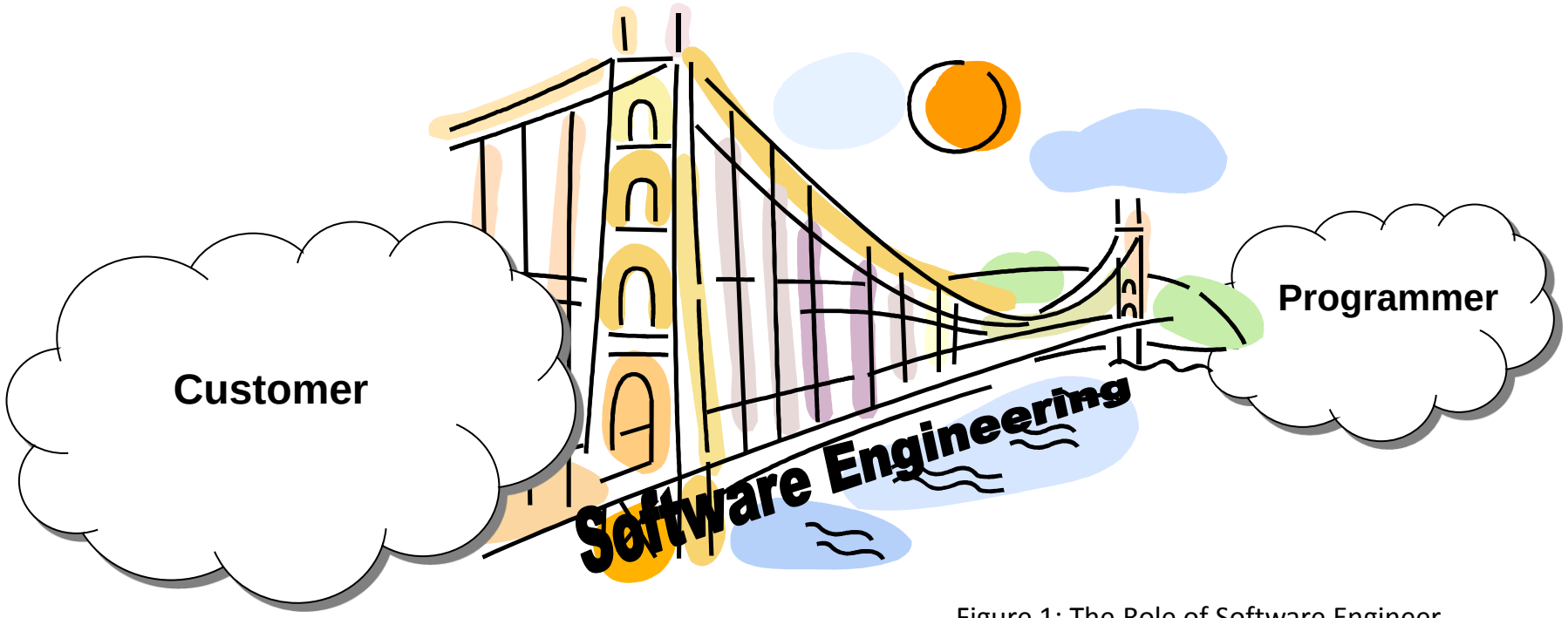


Figure 1: The Role of Software Engineer

First law of software engineering

Software engineer is willing to learn the problem domain (problem cannot be solved without understanding it first)

Role of a software engineer (2)

- Software engineering ***delivers*** value to the customer

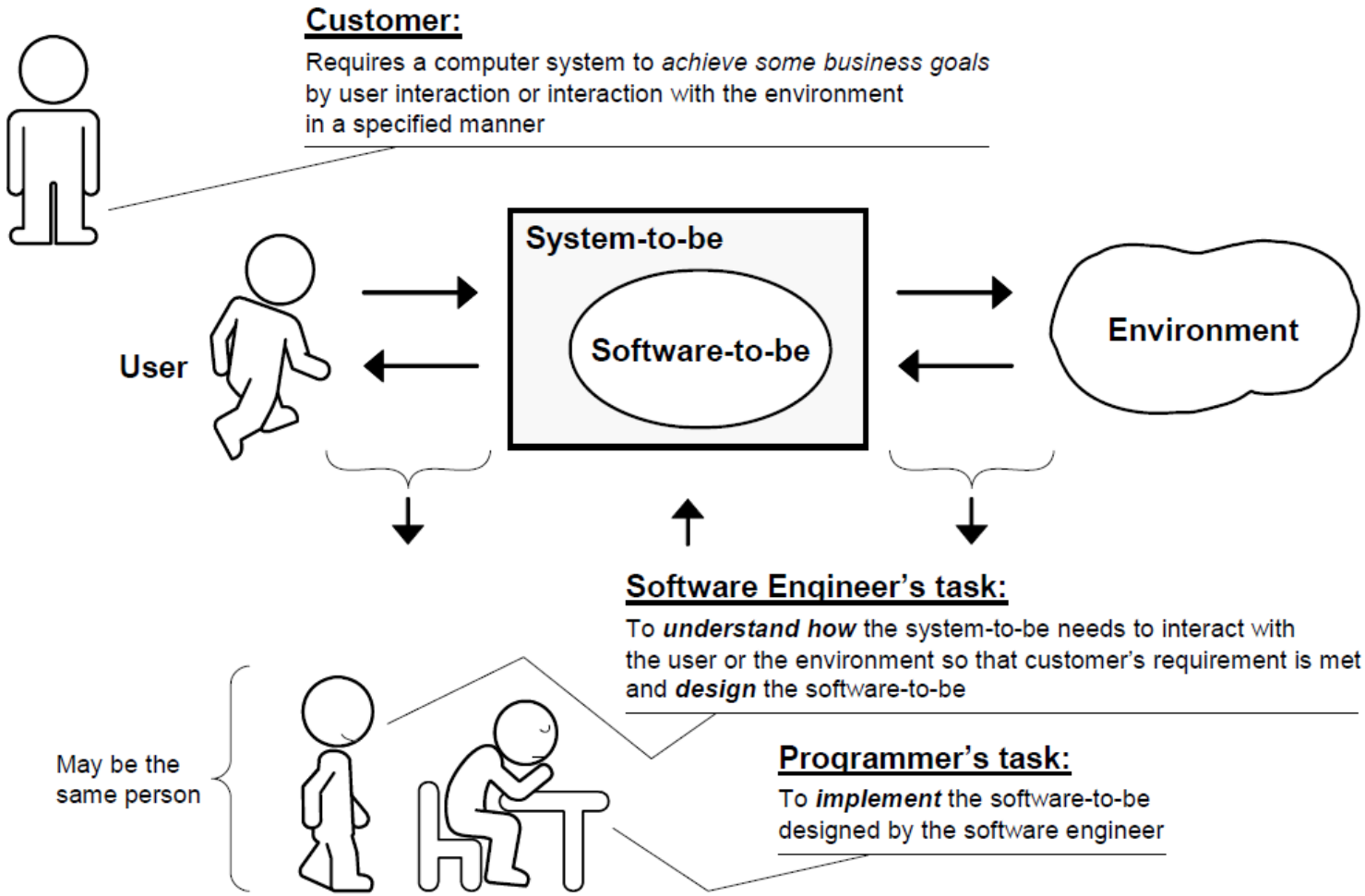
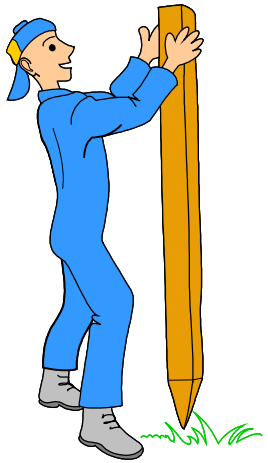


Figure 1: Role of Software Engineering

Why is software engineering complex (1) ?

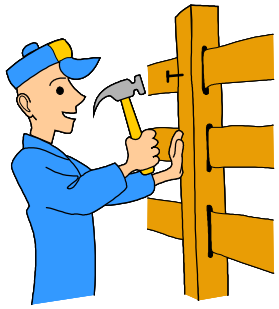
1. Software Development is Complex
   - Complex $\neq$ complicated
   - Complex = composed of many simple parts that are related to one another
   - Complicated = not well understood, or explained

Complexity Example:
Scheduling Fence Construction Tasks

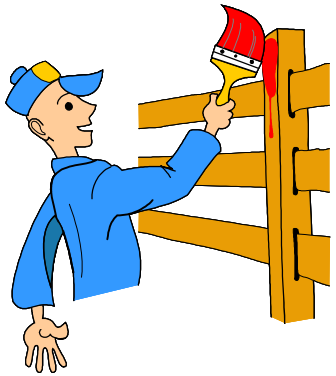


Setting posts
[3 time units]

Cutting wood
[2 time units]



Nailing
[2 time units for unpainted;
3 time units otherwise]



Painting
[5 time units for uncut wood;
4 time units otherwise]

Setting posts $<$ Nailing, Painting

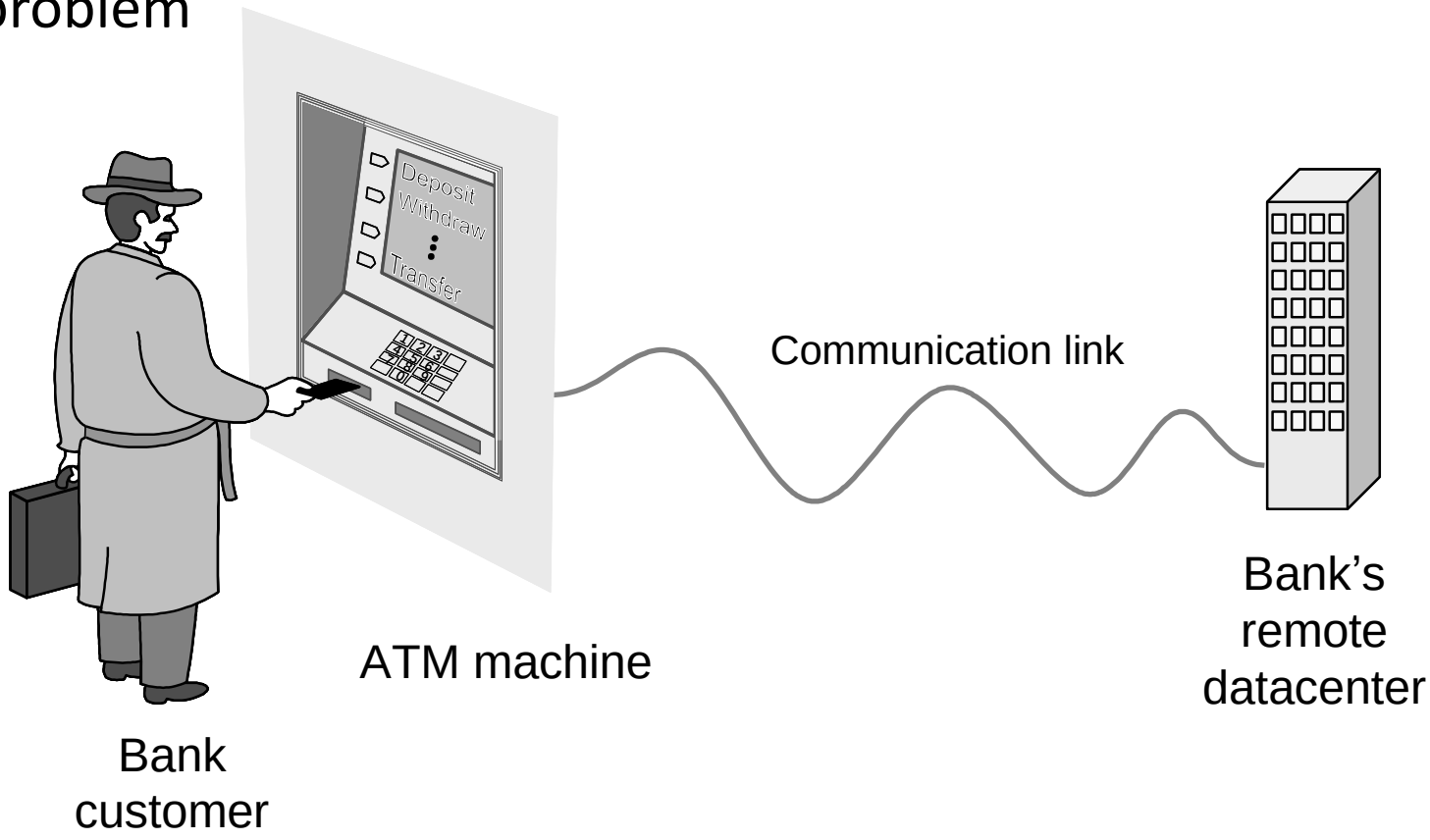
Cutting $<$ Nailing

…shortest possible completion time = ?

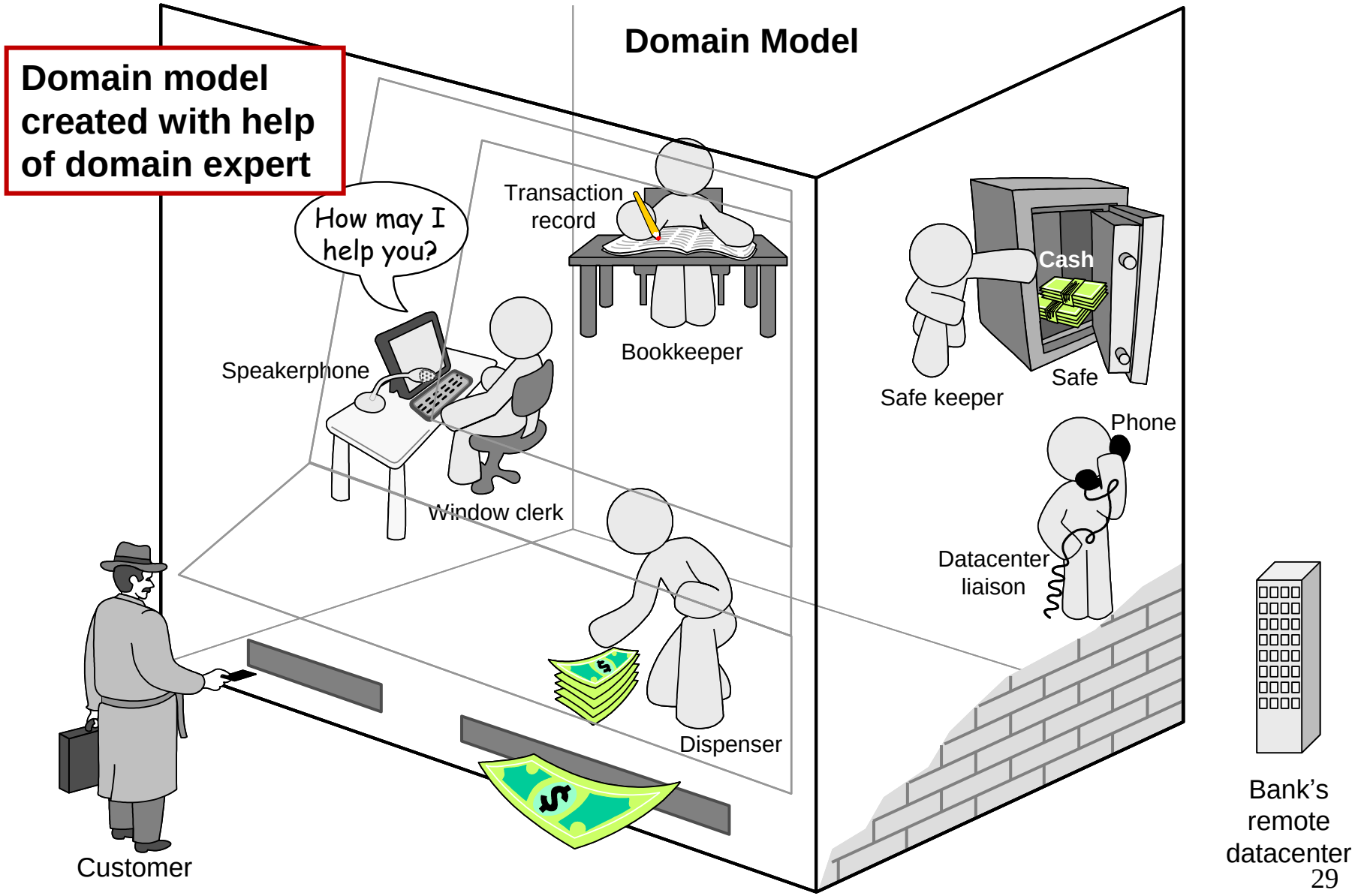
[$\Rightarrow$ “simple” problem, but hard to solve without a pen and paper]

Why is software engineering difficult(2) ?

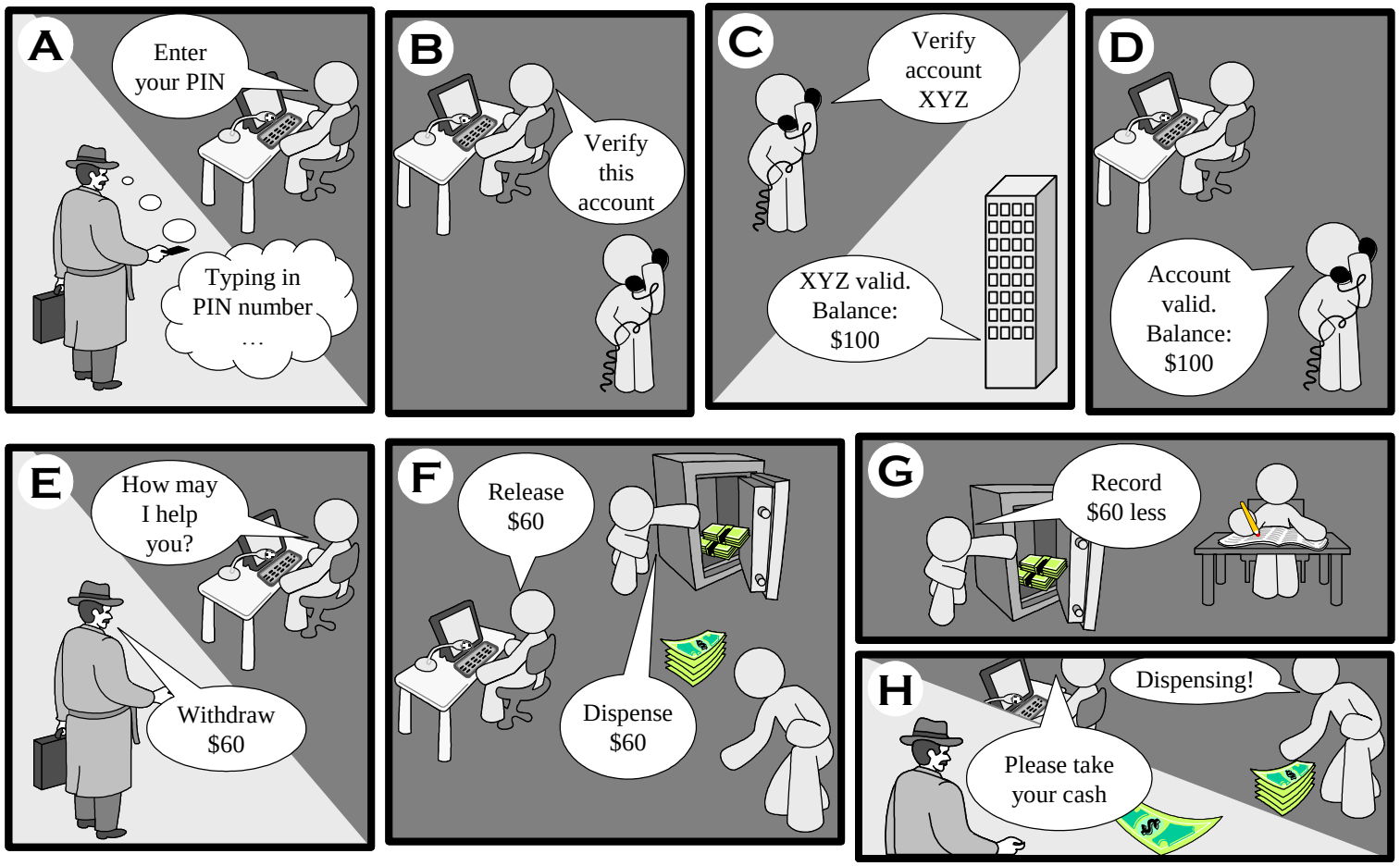
2. Software Engineering requires imagination
   e.g., ATM Machine – understanding the money-machine problem



How ATM Machine Might Work



Cartoon Strip: How ATM Machine Works



Why is software engineering difficult (3) ?

- As a software engineer:
  - You need to understand the *software domain* (that is what you are building)
  - You need to understand the *problem domain* (because that is what you are building a solution for)
  - Besides, software is a *formal domain* with well-defined inputs and goal, but real world is *informal* with ill-defined inputs and goal states
  - Challenge is to find a set of *good abstractions* that is representative of the problem domain
  - But, we live in a changing world…so *good abstractions* wear out, break and get dispersed

Software Engineering Life-Cycle

- We described software engineering as a complex, organised process with a great emphasis on *methodology*. This organised process can be broken into the following phases:
  - Analysis and Specification
  - Design
  - Implementation
  - Testing
  - Release & Maintenance
- Each of the above phases can be accompanied by an artifact or deliverable to be achieved at the completion of this phase
- The life-cycle usually comprises peripheral activities such as feasibility studies, software maintenance, software configuration management etc.

Software Engineering Life-Cycle

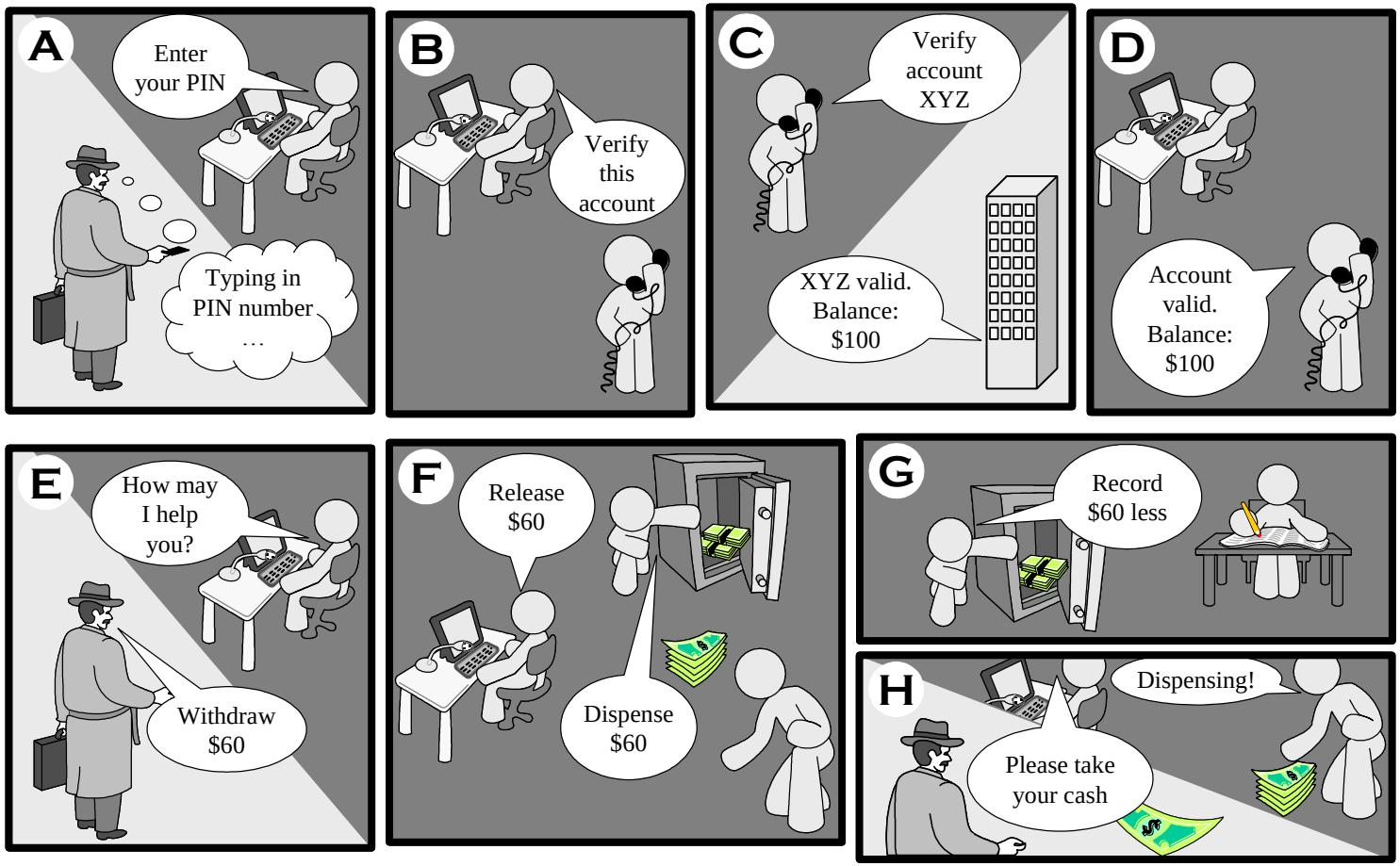
1. Analysis and Specification:
   - A process of knowledge-discovery about the "system-to-be" and list of features, where software engineers need to:
     - understand the problem definition (delimit its scope, elaborate the system's services (*behavioural characteristics)*
     - abstract the problem to define a domain model (*structural characteristics*)
   - Comprises both functional (inputs and outputs) and non-functional requirements (performance, security, quality, maintainability, extensibility)
   - Popular techniques include use-case modelling, user-stories…
2. Design:
   - A problem-solving activity that involves a "Creative process of searching **how to implement all of the customer's requirements"** and generating software engineering blue-prints

Software Engineering Blueprints

- As part of design process, essential to communicate your ideas
- When describing a process, people often tend to use abbreviations and symbols
- Miller (1957) believes, people use higher levels of abstraction to remember things: *chunking*
- Symbols can be easier chunked into patterns, represented into new symbols
- Using symbols and hierarchical abstractions makes it easier for people to think about complex systems
- Specifying software problems and solutions is like cartoon strip writing, unfortunately, most of us are not artists, so software engineers tend use something less exciting: UML symbols (class diagram, component diagram, deployment diagram – different symbols for different views)

Software Engineering Life-Cycle

- In generating these “blue-prints”, “cartoon strip writing” is fun, represents problem & solutions in a more comprehensible manner, but not formal enough



Cartoon Strip: How ATM Machine Works

Software Engineering Life-Cycle

3. Implementation:
   - The process of encoding the design in a programming language to deliver a software product
4. Testing:
   - A process of verification that our system works correctly and realises the goals
   - Testing process encompasses unit tests (individual components are tested), integration tests (the whole system is testing), user acceptance tests (the system achieves the customer requirements)
5. Operation & Maintenance:
   - Running the system; Fixing defects, adding new funtionality

Software Engineering Life-Cycle

- However, software development is unlike any other product development in these aspects:
  - software is *intangible* and hard to visualize.
  - software is probably the most *complex* artifact—a large software product consists of so many bits and pieces as well as their relationships
  - software is probably the most flexible artifact—it can be easily and radically modified at any stage of the development process, so it can quickly respond to changes in customer requirements Hence, a linear order of understanding the problem, designing a solution, implementing and deploying the solution, does not produce best results.
- It is easier to understand a complex problem by implementing and evaluating pilot solutions.

Incremental and Iterative Methods

- These insights led to adopting ***incremental*** and ***iterative*** development methods, which are characterized by:
  1. Break the big problem down into smaller pieces (increments) and prioritize them.
  2. In each iteration progress through the development in more depth.
  3. Seek the customer feedback and change course based on improved understanding.
- An incremental and iterative process
  - seeks to get to a working instance as soon as possible.
  - progressively deepen the understanding or "visualization" of the target product, by both advancing and retracting to earlier activities to rediscover more of its features.

COMP 1531
Software Engineering Fundamentals

Week 01: Wednesday

Software Development Methods

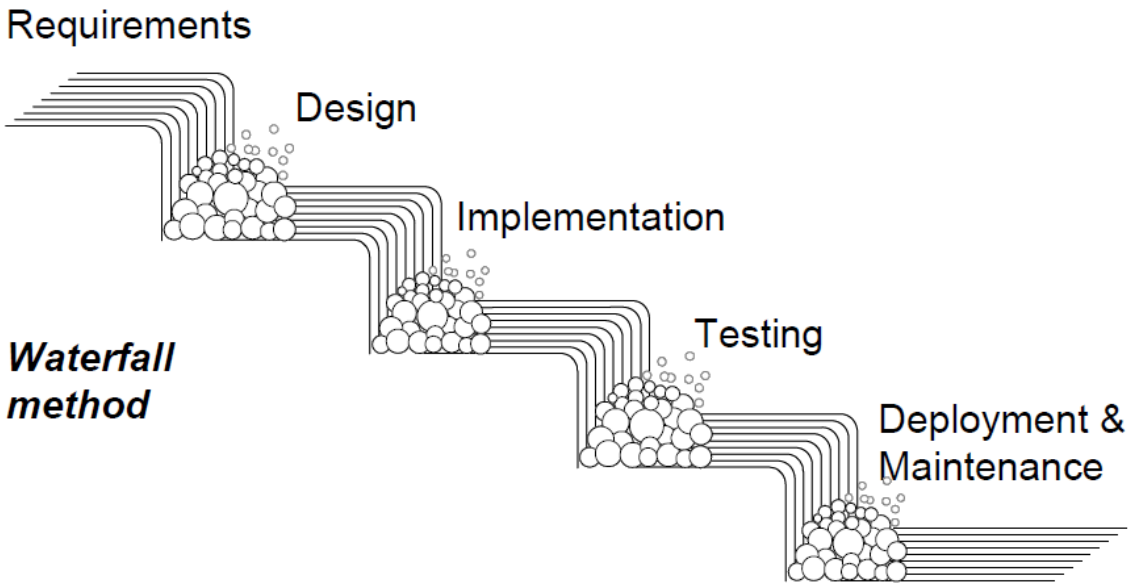
Aarthi Natarajan

Software Development Methodologies

- Method = work strategy
  - The Feynman Problem-Solving Algorithm: (i) Write down the problem (ii) think very hard, and (iii) write down the answer.
- A **software development method** lays out a prescriptive process by mandating a *sequence of development tasks*
- *Elaborate processes* with rigid, plan-driven, documentation heavy methodologies
  - Waterfall: Unidirectional, finish this step before moving to the next
- *Iterative & Incremental processes* which develop increments of functionality and repeat in a feedback loop
  - Rational Unified Process [Jacobson et al., 1999]
  - Agile methods (e.g., SCRUM, XP):
    - Methods that are more aggressive in terms of short iterations
    - Heavy customer involvement, user feedback essential; feedback loops on several levels of granularity; customer is continuously asked to prioritize the remaining work items and provide feedback about the delivered increments of software.

Waterfall Model (1970's)

- Traditional, linear, sequential life cycle model also known as *plan-driven development model* with **detailed planning**
  - Detailed planning – problem is identified, documented and designed
  - Implementation tasks identified, scoped and scheduled
  - Development cycle followed by testing cycle
- Simple to understand and manage due to *project visibility* i.e. better control over all the processes in the project because of clear visibility in all the phases
- Suitable for simple, risk-free projects with **stable** product statement, clear, well-known requirements with no ambiguities, technical requirements clear and resources ample or mission-critical applications (e.g., NASA)

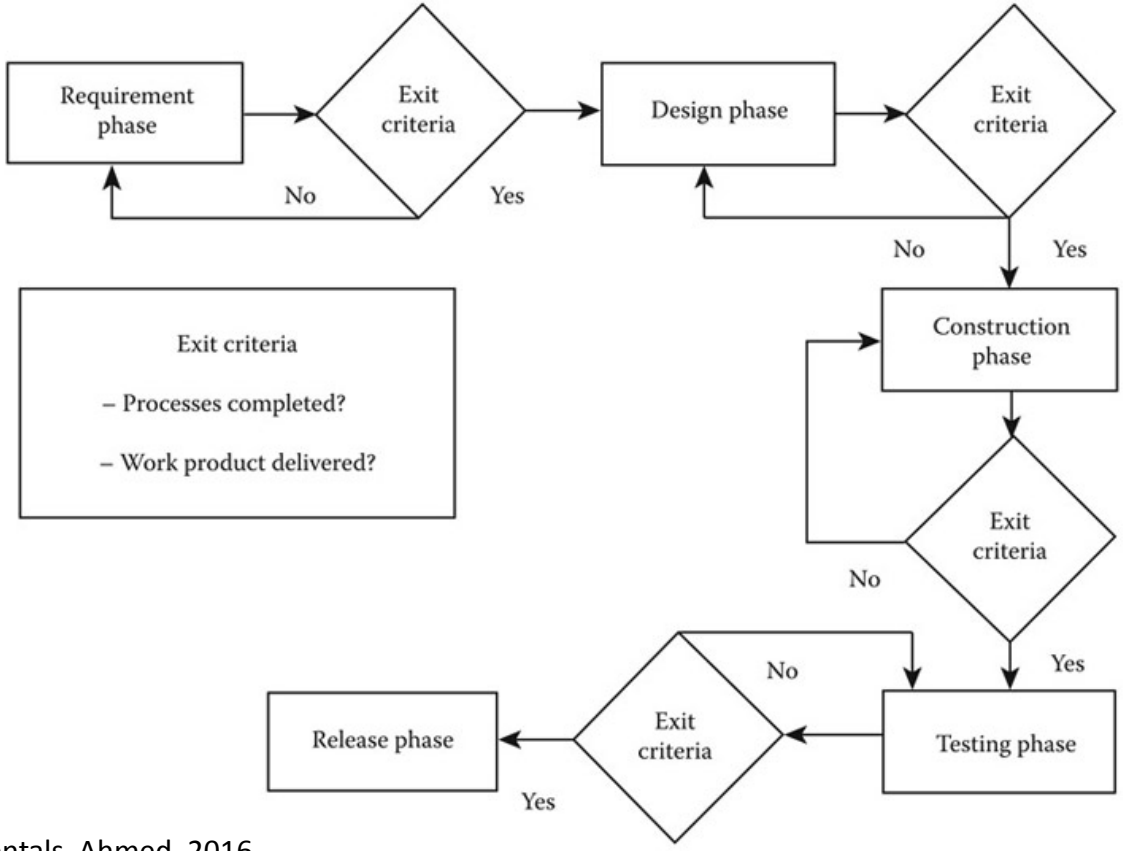


Waterfall process for software development.

Waterfall Model Variation

A **waterfall model variation** that implements a quality gate system

- A process model that ensures that quality is maintained throughout the life-cycle
- A ***completion criteria check*** is done that does a quality assurance check to see if all the necessary artifacts are generated for that phase and the artifacts meet the quality standards



Waterfall Model Drawbacks

- No working software is produced until late into the software life-cycle
- Rigid and not very flexible
  - Does not support fine-tuning or refinement of customer requirements through the cycle
  - Good ideas need to be identified upfront
  - As typically all requirements are frozen at the end of the requirements phase, once the application is implemented and in the "testing" phase, it is difficult to retract and change something that was not "well-thought out" in the concept phase or design phase
  - A great idea in the release cycle is a threat
- Heavy documentation (typically model based artifacts, UML)
- Not suitable for projects where requirements are at a moderate risk of changing
- Typically incurs a large management overhead

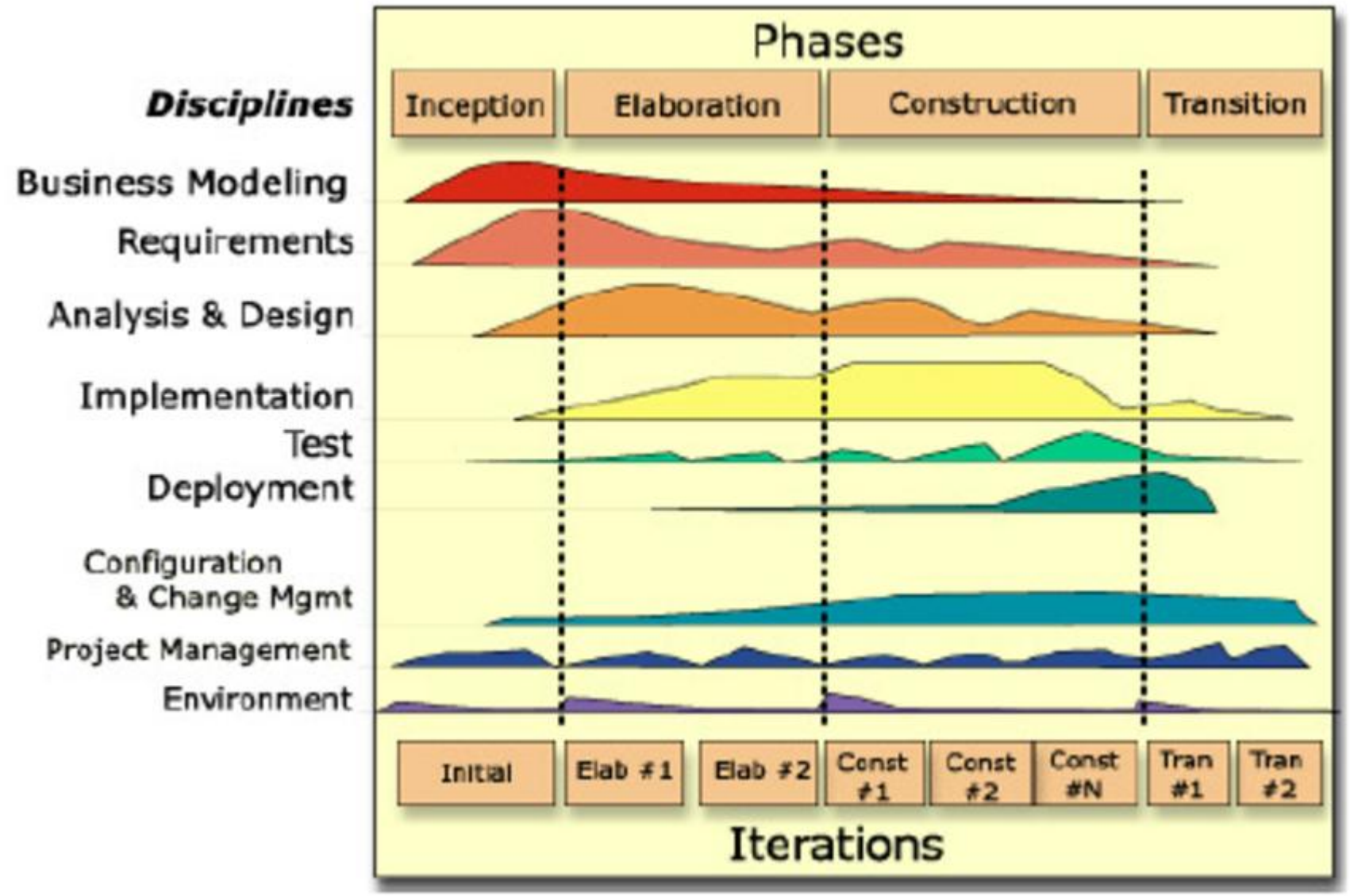
Rational Unified Process (RUP)

- An iterative software development process developed by Ivar Jacobson, Grady Booch and James Rumbaugh which has a series of four phases:
  - **Inception** – scope the project, identify major players, what resources are required, architecture and risks, estimate costs
  - **Elaboration –** understand problem domain, analysis, evaluate in detail required architecture and resources
  - **Construction –** design, build and test software
  - **Transition –** release software to production
- RUP is ***serial in the large*** and ***iterative in the small***
  - The four phases occur in a serial manner over time (hmm…sounds like sequential), however…
  - Work in an iterative manner on a day-to-day basis (some modelling, some implementation, some testing, some management…)

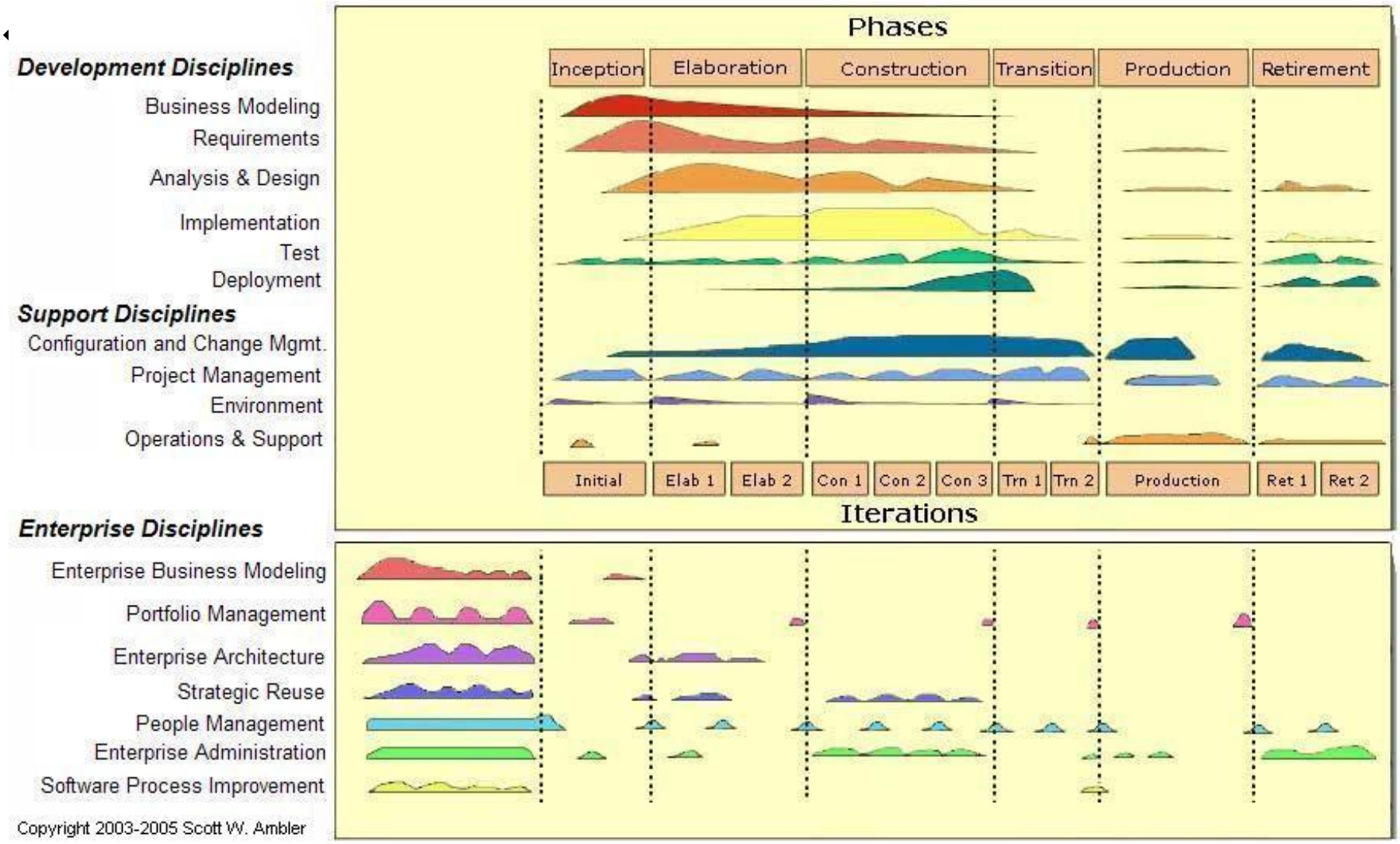
Rational Unified Process (RUP)

- All work in RUP organised into ***disciplines*** (previously workflows) :
  - **Development disciplines**
    - Business Modelling: understanding domain, develop a high-level requirements model (use-case model)
    - Requirements: Identify, model and document vision and requirements (use-case model, , domain model (class or data diagram), a business process model (a data flow diagram, activity diagram)
    - Analysis & Design: Engineer the blue-print
    - Implementation: Encode the design
    - Test: Testing throughout the project
    - Deployment: Product releases, software delivery
  - **Support disciplines**
    - Configuration and Change Management, Project Management, Environment
- **UP is not inherently documentation centric**.

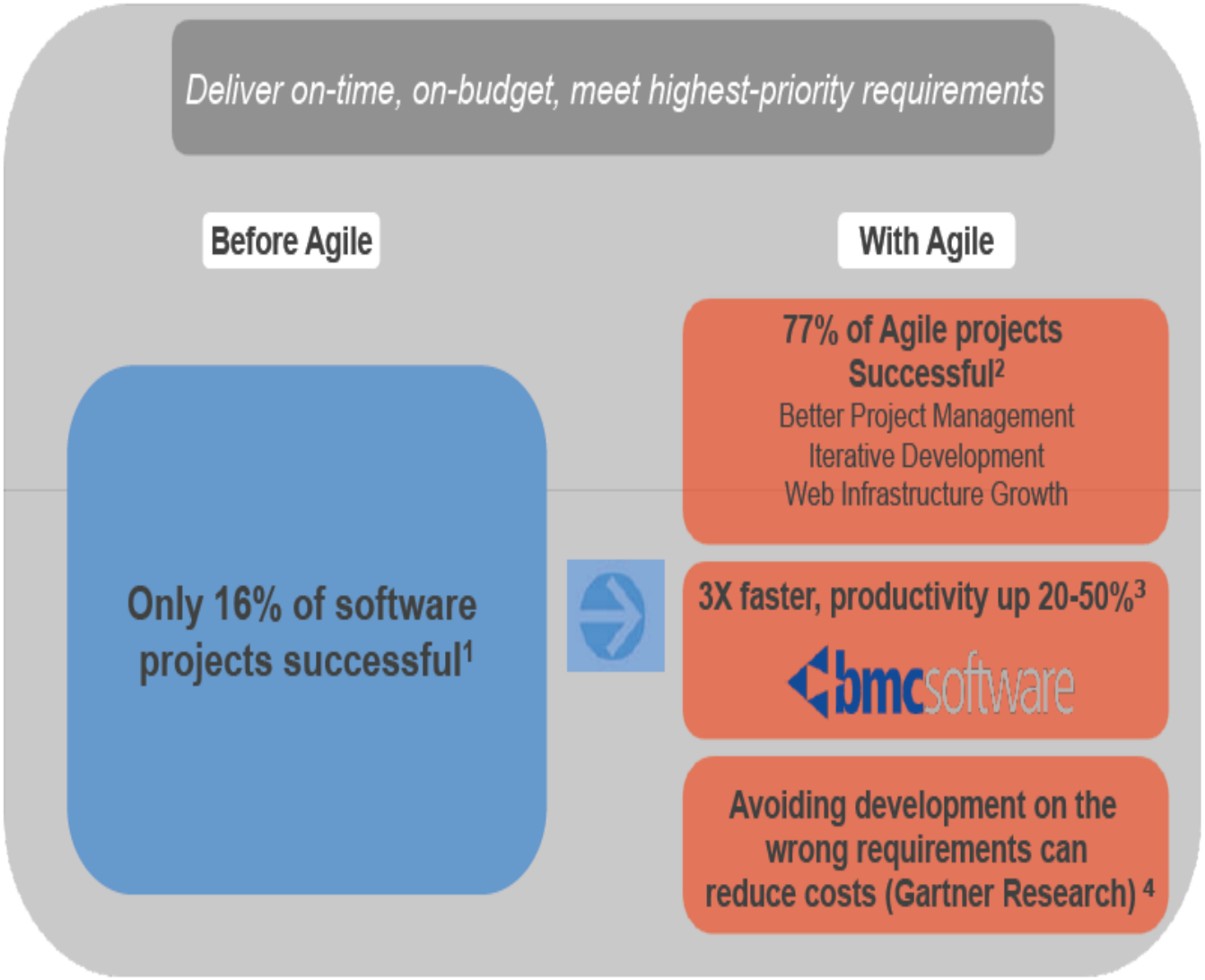
Rational Unified Process (RUP)

- 

Enterprise Unified Process (EUP)



Why Do We Need Agile?



1 Standish Group Report: There's Less Development Chaos Today, by David Rubinstein *SD Times March 1,* 2007, **2 "Agile Has Crossed the Chasm," *Dr. Dobb's Journal, July 2, 2007. 3QSMA and Cutter Consortium ROI***
case study on BMC Software, 2008. **4 Gartner, Inc. 2005**
3 Why agile - Rally software development corp

Why Do We Need Agile?

- **93% increased productivity**
- **88% increased quality**
- **83% improved stakeholder** satisfaction
- **49% reduced costs**
- **66% three-year, risk-adjusted return** on investment

1 “Agile Methodologies: Survey Results,” by Shine Technologies, 2003; 2 Forrester Research, 2004;

Agile Manifesto (Agile Alliance, 2001)

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

12 principles of Agile Software

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development.
  The sponsors, developers, and users should be able
  to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs
  emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile drawbacks

- Daily stand up meetings and close collaboration makes it difficult to adapt the process to development outsourcing, clients and developers separated geographically, or business clients who simply don't have the manpower, resources or interest to spare.
- emphasis on modularity, incremental development, and adaptability doesn't suit it easily to clients who want contracts with firm estimates and timetables
- Its reliance on small self-organized teams makes it difficult to adapt to large software projects with many stakeholders with different needs and neglects to take into account the need for leadership while team members get used to working together.
- Further, lack of comprehensive documentation can make it difficult to maintain or add to the software after members of the original team turn over, and can lead to modules with inconsistent features and interfaces.
- Lastly, more than with Waterfall and RUP, Agile development typically depends on the ability to recruit very experienced software engineers who know how to work independently and interface effectively with business users.

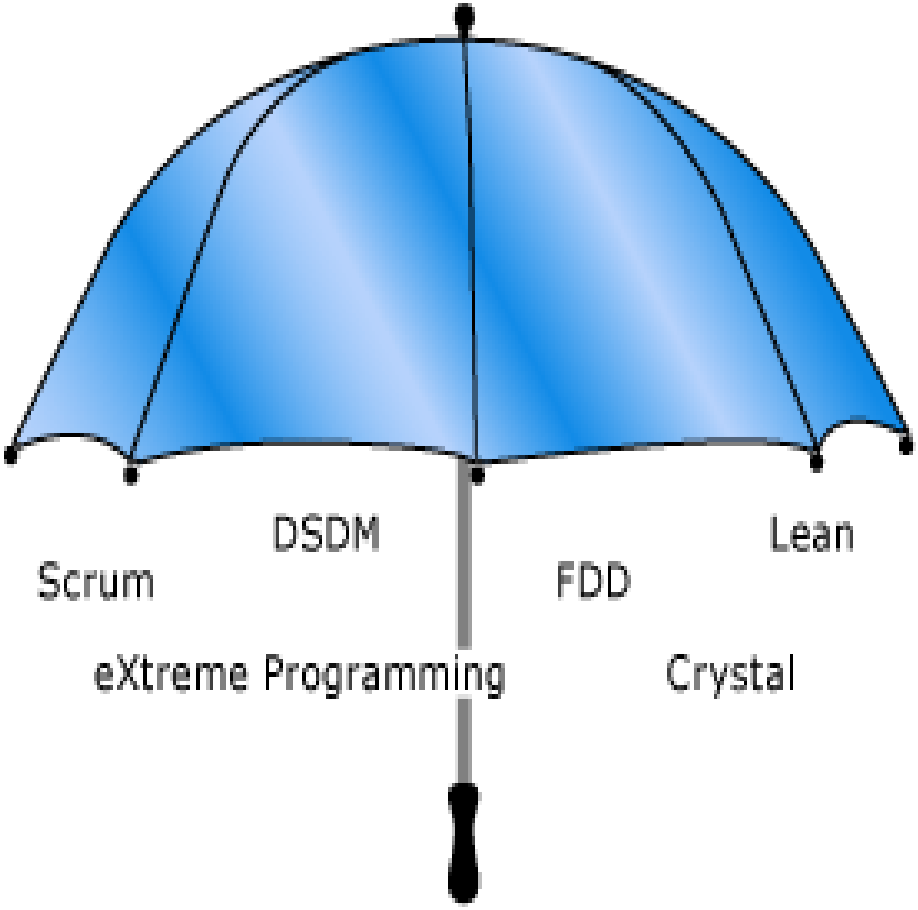
Which methodology?

- What does the customer want?

  need software yesterday with the most advanced features at the lowest possible cost !

- No one methodology is the best fit
- secret to successful software development is to understand all three processes in depth and take the parts of each that are most suited to your particular product and environment.
- Stay agile in your approach through constant re-evaluation and revising the development process
- SaaS (Software as a Service) and Web 2.0 applications that require moderate adaptability are likely to be suited to agile style
- Mission-critical applications such as military, medical that require a high degree of predictability are more suited to waterfall

Agile Software Development Methodologies



Useful Links

- http://www.agilemodeling.com/essays/agileModelingRUP.htm
- http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf (Chapter 1 - pages 1- 31)
- https://docs.python.org/3/tutorial/
- https://learnxinyminutes.com/docs/python3/