Using Simulation to teach project management skills

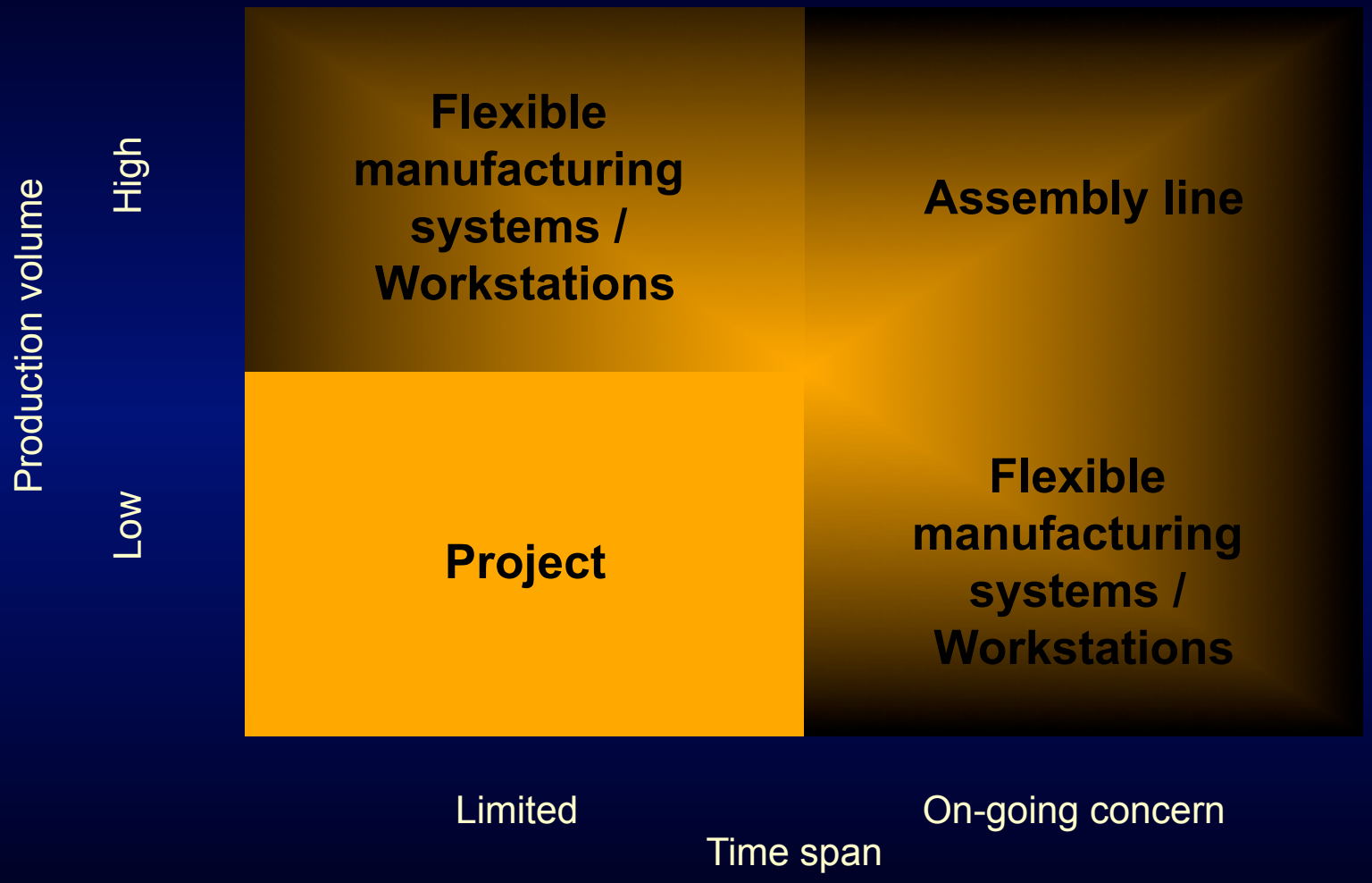
Dr. Alain April, ÉTS Montréal
alain.april@etsmtl.ca

Agenda of the workshop

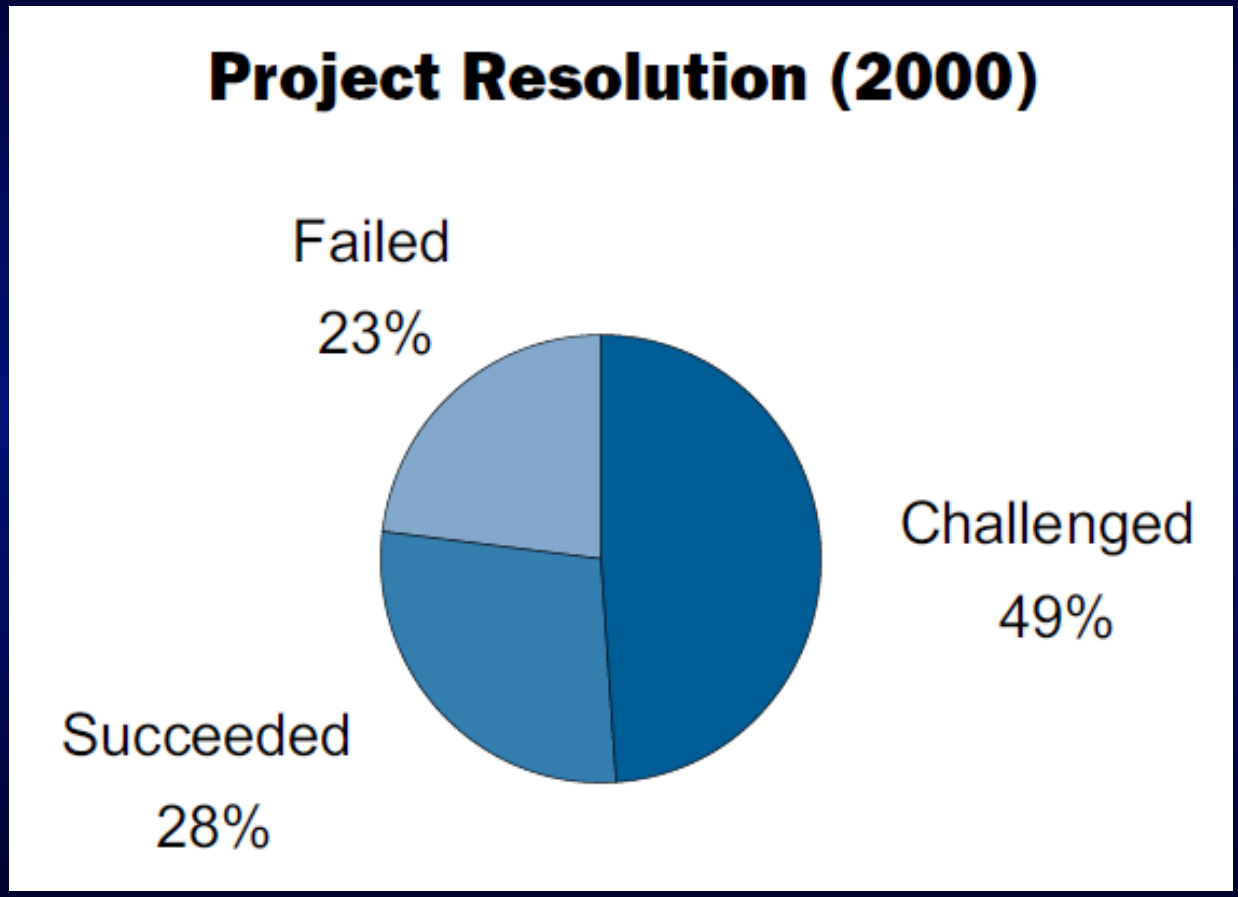
1. The software project management theory overview (40 minutes)
2. Why use SDLC simulation with students (2 minutes)
3. Introduction to the SIMSE simulator (5 minutes)
4. SIMSE installation and interface overview
5. SIMSE simulator execution & support tools
6. SIMSE examples of student results
7. Questions and Answers

1. The software project management theory overview (30 minutes)

A project is a form of organizing work



Many projects: 1) Do not deliver on-time, 2) Do not deliver on budget, 3) Do not deliver

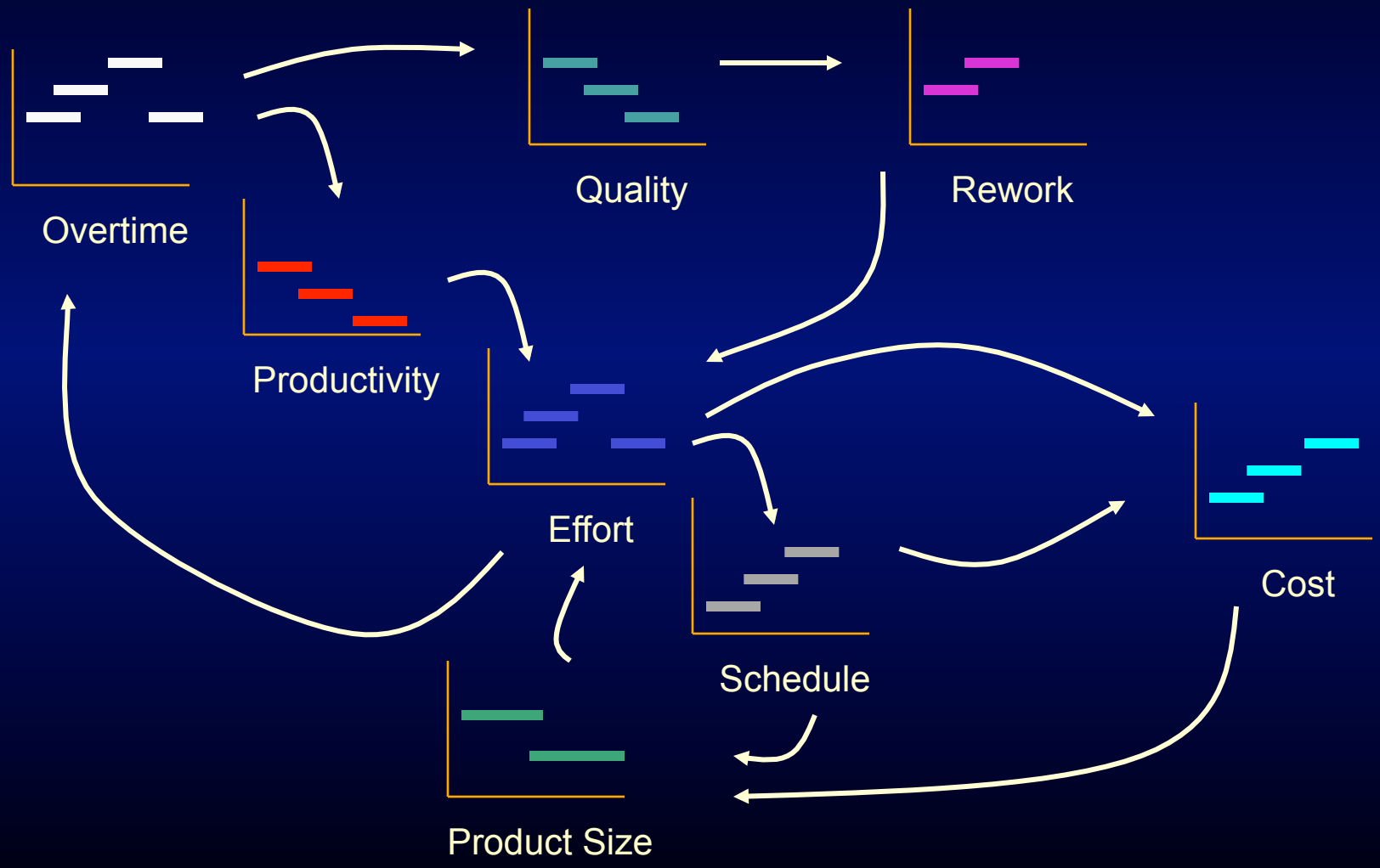


Chaos Report, Standish Group, 2001

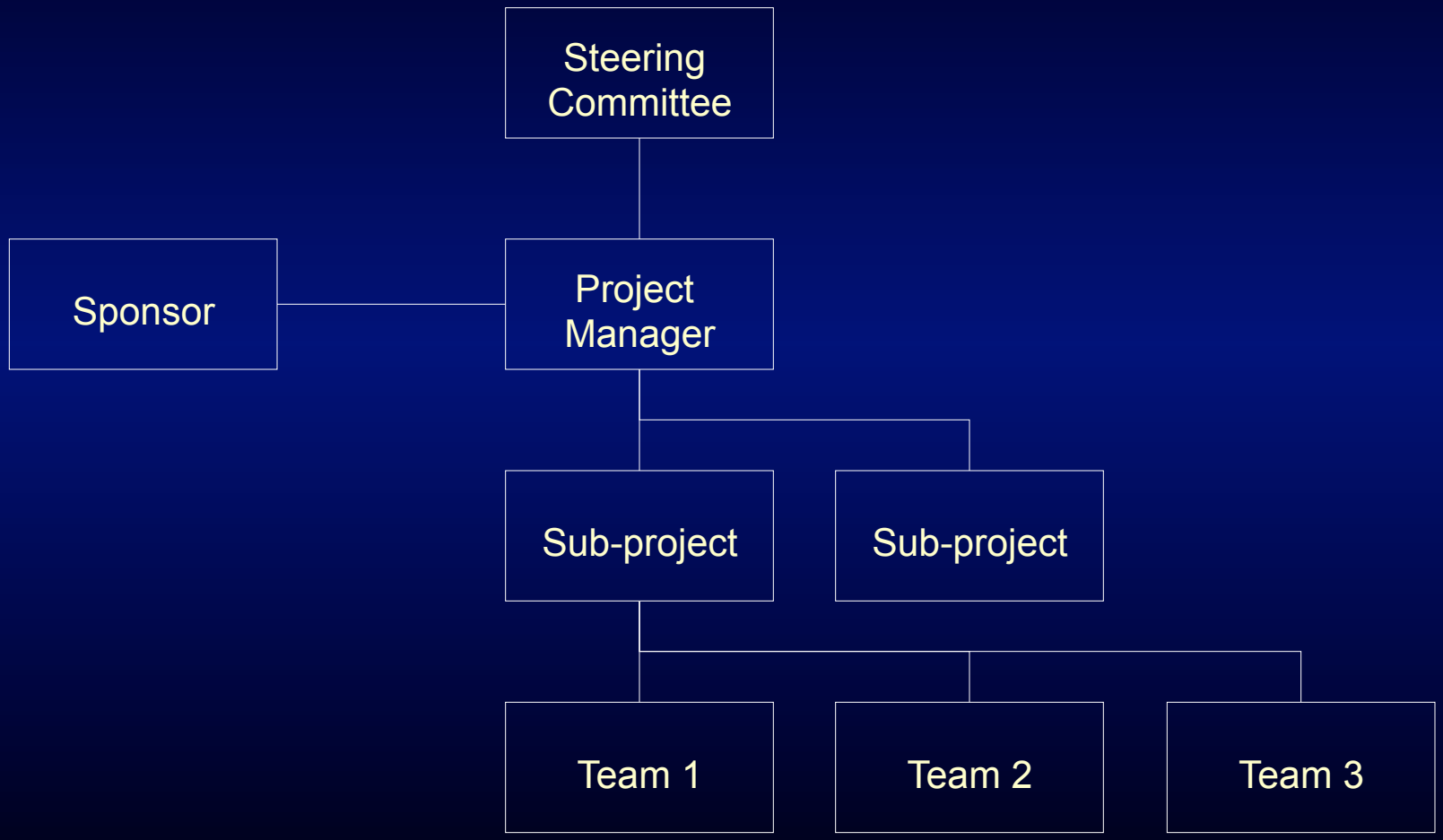
Projects have complex behaviours

- They can not be steered in the desired direction by doing just one thing at a time, be it overtime, reducing scope or increasing one project's headcount, and that there is an omnipresent present risk of "over steering" the project.
- Responses to the actions of the project manager are non-linear, time-lagged and time dependent. The demonstrability of causes and effects are ambiguous. Inputs and outputs are not proportional. The whole is not quantitatively equal to its parts, or even qualitatively recognizable in its constituent components.

Project Management Dynamics



Roles in the Project Organization



The project management work

- What?
  - Identify work to be done
  - Produce estimates
  - Plan work, resources & funds
  - Identify issues & risks
  - Acquire Resources
  - Establish performance baseline
  - Measure progress
  - Take corrective action
  - Motivate
  - Communicate
- When
  - At the beginning of the project
  - Rolling wave

- How
  - WBS
  - OBS
  - RBS
  - CBS
  - PERT
  - CCM
  - EVM
  - FP
  - SDLC
  - SLOC
  - LOB
  - WP
  - PMB
  - …

System Development Life Cycle (SDLC)

- Project
  - a planned undertaking that has a beginning and an end, and which produces a predetermined result or product.
- Systems development project
  - Planned undertaking
  - Large job
  - Produces new system (many artifacts + source code)
- Successful project requirements
  - Detailed plans
  - Organized, methodical sequence of tasks and activities

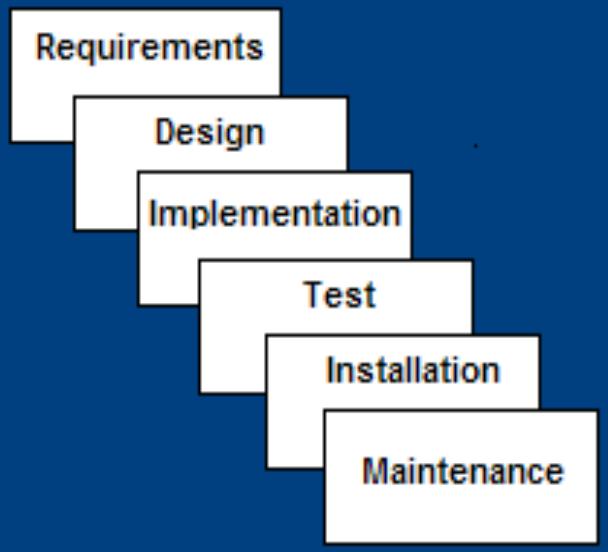
Limited resources and identifying work

- A Project has limited resources and constraints
  - A complete and error free software must be produced by a team of employees within a certain schedule (clicks in simulations) and budget
- Producing a software (Interacting with employees)
  - Requires the project manager to intelligently assign employees on activities of the SDLC
  - All employees are paid so they must be assigned to work at all times
  - Using SE tools will help produce software faster with lower amount of defects
  - Employees have specific experience, a salary rate, Energy and Mood levels which impact their productivity
  - When employees work they also generate hidden errors that will need to be removed by assigning employees to review the artifacts = finding errors to remove them

Choosing a SDLC will influence project management practices

- SDLC: is a framework that describes the activities performed at each stage of a software development project;
- There are many types to choose from:
  - The Waterfall model;
  - The incremental model;
  - The rapid prototyping model;
  - Agile SDLC : The Extreme Programming model;
  - The Rational Unified model.

Waterfall Model



- **Requirements Documents** – defines needed information, function, behavior, performance and interfaces.
- **Design Documents** – data structures, software architecture, interface representations, algorithmic details.
- **Source Code** – database, user documentation.
- **Testing** – System test plan, test cases, error reports, re-test

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

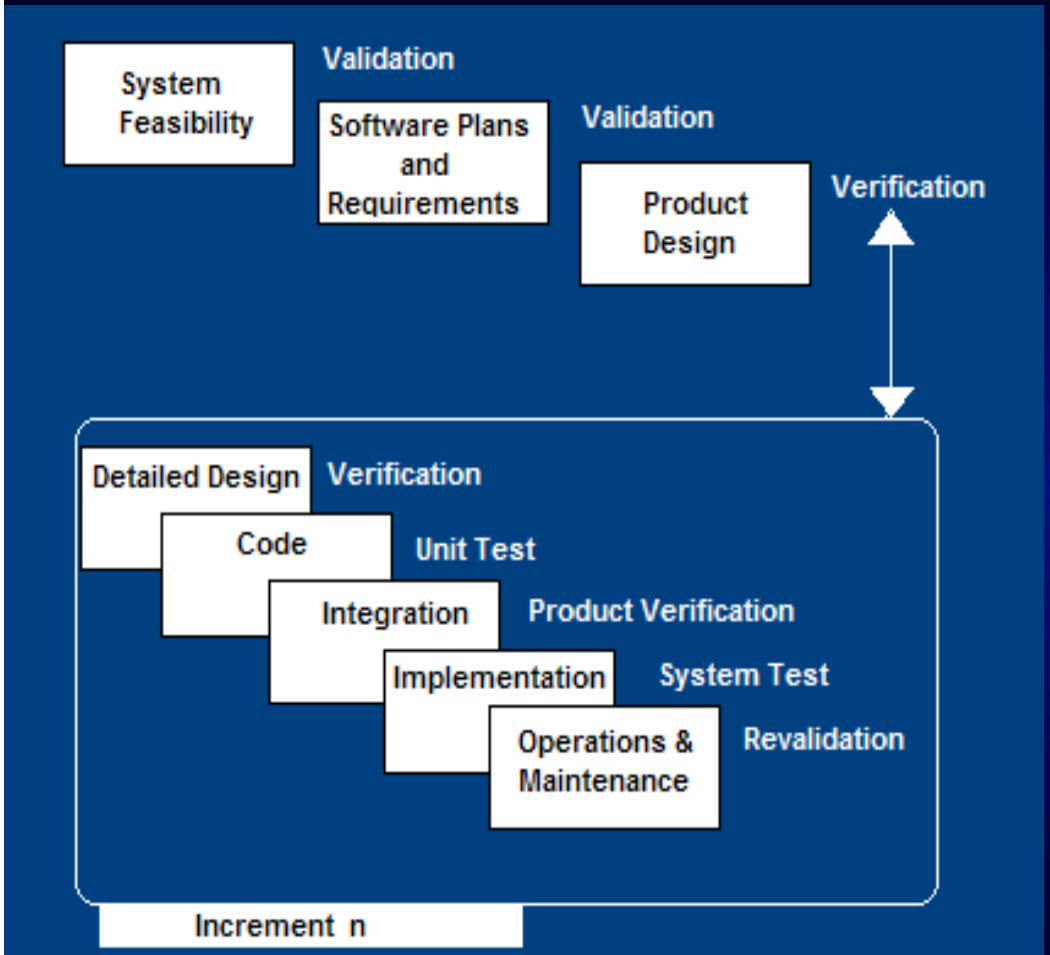
Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

Incremental SDLC Model



- Construct a partial implementation of a total system
- Then slowly add increased functionality
- The incremental model prioritizes requirements of the system and then implements them in groups.
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

Incremental Model Strengths

- Develop high-risk or major functions first
- Each release delivers an operational product
- Customer can respond to each build
- Uses "divide and conquer" breakdown of tasks
- Lowers initial delivery cost
- Initial product delivery is faster
- Customers get important functionality early
- Risk of changing requirements is reduced

Incremental Model Weaknesses

- Requires good planning and design
- Requires early definition of a complete and fully functional system to allow for the definition of increments
- Well-defined module interfaces are required (some will be developed long before others)
- Total cost of the complete system is not lower

When to use the Incremental Model

- Risk, funding, schedule, program complexity, or need for early realization of benefits.
- Most of the requirements are known up-front but are expected to evolve over time
- A need to get basic functionality to the market early
- On projects which have lengthy development schedules
- On a project with new technology

Rapid Application Model (RAD)

- Requirements planning phase (a workshop utilizing structured discussion of business problems)
- User description phase – automated tools capture information from users
- Construction phase – productivity tools, such as code generators, screen generators, etc. inside a time-box. (“Do until done”)
- Cutover phase -- installation of the system, user acceptance testing and user training

RAD Strengths

- Reduced cycle time and improved productivity with fewer people means lower costs
- Time-box approach mitigates cost and schedule risk
- Customer involved throughout the complete cycle minimizes risk of not achieving customer satisfaction and business needs
- Focus moves from documentation to code (WYSIWYG).
- Uses modeling concepts to capture information about business, data, and processes.

RAD Weaknesses

- Accelerated development process must give quick responses to the user
- Risk of never achieving closure
- Hard to use with legacy systems
- Requires a system that can be modularized
- Developers and customers must be committed to rapid-fire activities in an abbreviated time frame.

When to use RAD

- Reasonably well-known requirements
- User involved throughout the life cycle
- Project can be time-boxed
- Functionality delivered in increments
- High performance not required
- Low technical risks
- System can be modularized

Agile SDLC's

- Speed up or bypass one or more life cycle phases
- Usually less formal and reduced scope
- Used for time-critical applications
- Used in organizations that employ disciplined methods

Some Agile Methods

- Adaptive Software Development (ASD)
- Feature Driven Development (FDD)
- Crystal Clear
- Dynamic Software Development Method (DSDM)
- Rapid Application Development (RAD)
- Scrum
- Extreme Programming (XP)
- Rational Unify Process (RUP)

Extreme Programming – (XP)

For small-to-medium-sized teams developing software with vague or rapidly changing requirements

Coding is the key activity throughout a software project

- Communication among teammates is done with code
- Life cycle and behavior of complex objects defined in test cases – again in code

XP Practices (1-6)

1. Planning game – determine scope of the next release by combining business priorities and technical estimates
2. Small releases – put a simple system into production, then release new versions in very short cycle
3. Metaphor – all development is guided by a simple shared story of how the whole system works
4. Simple design – system is designed as simply as possible (extra complexity removed as soon as found)
5. Testing – programmers continuously write unit tests; customers write tests for features
6. Refactoring – programmers continuously restructure the system without changing its behavior to remove duplication and simplify

XP Practices (7 – 12)

7. Pair-programming -- all production code is written with two programmers at one machine
8. Collective ownership – anyone can change any code anywhere in the system at any time.
9. Continuous integration – integrate and build the system many times a day – every time a task is completed.
10. 40-hour week – work no more than 40 hours a week as a rule
11. On-site customer – a user is on the team and available full-time to answer questions
12. Coding standards – programmers write all code in accordance with rules emphasizing communication through the code

XP is “extreme” because

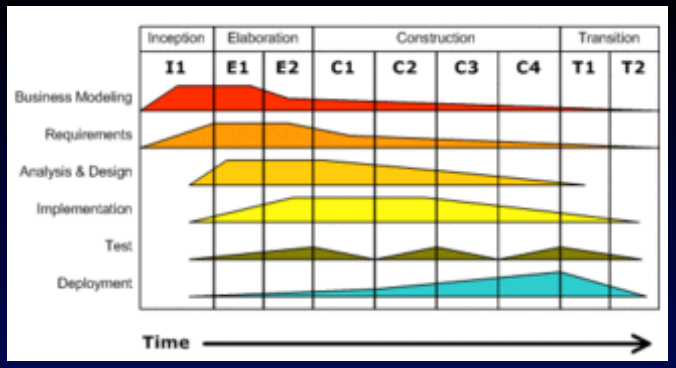
Commonsense practices taken to extreme levels

- If code reviews are good, review code all the time (pair programming)
- If testing is good, everybody will test all the time
- If simplicity is good, keep the system in the simplest design that supports its current functionality. (simplest thing that works)
- If design is good, everybody will design daily (refactoring)
- If architecture is important, everybody will work at defining and refining the architecture (metaphor)
- If integration testing is important, build and integrate test several times a day (continuous integration)
- If short iterations are good, make iterations really, really short (hours rather than weeks)

Rational Unified Processes model (RUP)

- In its simplest form, RUP consists of some fundamental workflows:
  - Business Engineering: Understanding the needs of the business.
  - Requirements: Translating business need into the behaviors of an automated system.
  - Analysis and Design: Translating requirements into software architecture.
  - Implementation: Creating software that fits within the architecture and has the required behaviors.
  - Test: Ensuring that the required behaviors are correct, and that all required behaviors are present.
  - Configuration and change management: Keeping track of all the different versions of all the work products.
  - Project Management: Managing schedules and resources.
  - Environment: Setting up and maintaining the development environment.
  - Deployment: Everything needed to roll out the project.

RUP model rules



Six Best Practices

1 - Develop iteratively

It is best to know all requirements in advance; however, often this is not the case. Several software development processes exist that deal with providing solution on how to minimize cost in terms of development phases.

2 - Manage requirements

Always keep in mind the requirements set by users.

3 - Use components

Breaking down an advanced project is not only suggested but in fact unavoidable.

RUP model rules

4 - Model visually

Use diagrams to represent all major components, users, and their interaction. "UML", short for Unified Modeling Language, is one tool that can be used to make this task more feasible.

5 - Verify quality

Always make testing a major part of the project at any point of time. Testing becomes heavier as the project progresses but should be a constant factor in any software product creation.

RUP model rules

6 - Control changes

Many projects are created by many teams, sometimes in various locations, different platforms may be used, etc. As a result it is essential to make sure that changes made to a system are synchronized and verified constantly. (See Continuous integration).

2. Why use Software Development Life-Cycle simulation with students

Why use Software Development Life-Cycle simulation with students

- Simulation allows students to experience how each SDLC impacts their management style;
- Iterative use of simulator allows them to test different assumptions and see the resulting score;
- A great way to experience/debate on the abstract concepts imbedded in software project management

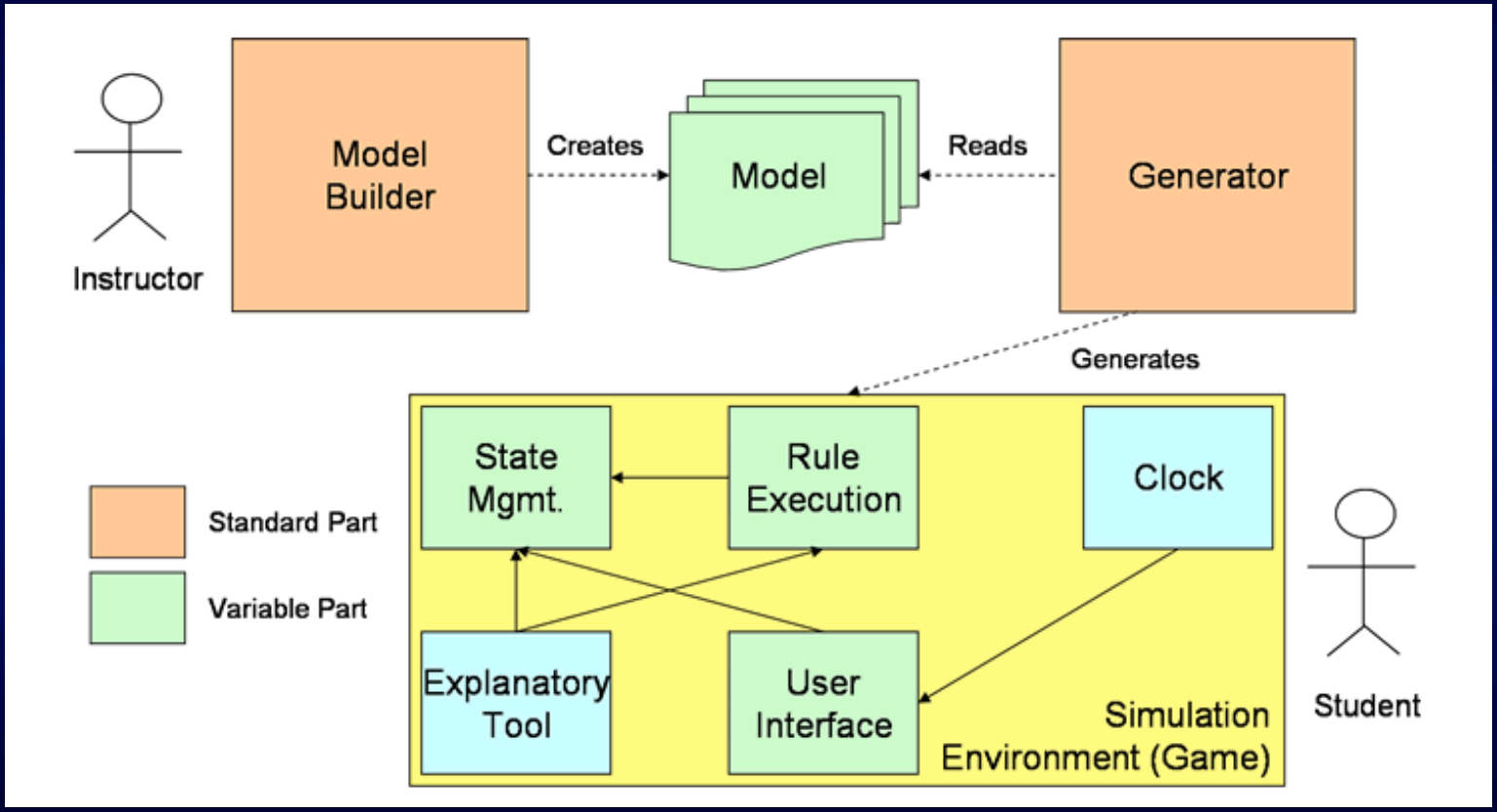
Introduction to the SIMSE simulator

An educational, Game-Based Software Engineering Simulation Environment by Emily Navarro of University of California - Irvine

click here

Introduction to SIMSE Simulator

- How is it built ?



Introduction to SIMSE Simulator

- How long is a typical simulation ?
  - 30 minutes to 2 hours for the Waterfall
  - 20 minutes to an hour for RUP
  - 10 to 20 minutes for RAD
- How do I use SIMSE in my class ?
  - The simplest way SimSE could be used is by just assigning students a simulation as a homework exercise. Of course, being the least involved approach, this may also provide the least inimal benefit of all of the approaches.
  - An instructor could also use SimSE in class, as an illustration tool for the concepts being taught in a lecture. For example, when talking about different life cycles, an instructor could show SimSE simulations of each life cycle model. Of course, this option would require more time and more effort in actually building different models, but would also be more rewarding than more simple options.

Introduction to SIMSE Simulator

- What do I need to run the simulator ?
  - The latest JDK (Java SE Development Kit)
- How do I get the SDLC I want to simulate ?
  - Each SDLC is ready to use, all you have to do is download it from the website http://www.ics.uci.edu/~emilyo/SimSE/downloads.html
- How do I start a game and how does it work ?
  - Once downloaded a game (.exe) you should read the .txt file which will explain your game objective. As an example the Waterfall objective is:
  - Your task is to create a a WEB system: Groceries@home
  - Your Budget: $280,000
  - Your timeline: $1,350 clock ticks
- Ready ? Then just click on the waterfall.exe to start the game

Introduction to SIMSE SImulator

Lets listen to a tutorial showing how:

- Beginning a game
- Viewing Resources (5 types: Artifacts, Customers, Employees, Projects and Tools)

click here

- How is SIMSE to be used in the classroom?
  - The simplest way SimSE could be used is by just assigning students a simulation as a homework exercise. Of course, being the least involved approach, this may also provide the least inimal benefit of all of the approaches.
  - An instructor could also use SimSE in class, as an illustration tool for the concepts being taught in a lecture. For example, when talking about different life cycles, an instructor could show SimSE simulations of each life cycle model. Of course, this option would require more time and more effort in actually building different models, but would also be more rewarding than more simple options. An additional enhancement to this approach would be having each student in the class also run the simulations on their laptops.
  - Ideally, reflection and dialogue sessions should accompany student use of SimSE, in which they are asked to reflect on and discuss the lessons they learned. This is also helpful in evaluating the quality of the models.
  - Advanced students could even build their own models of software processes.