Génie Logiciel et Gestion de Projets

Project Management

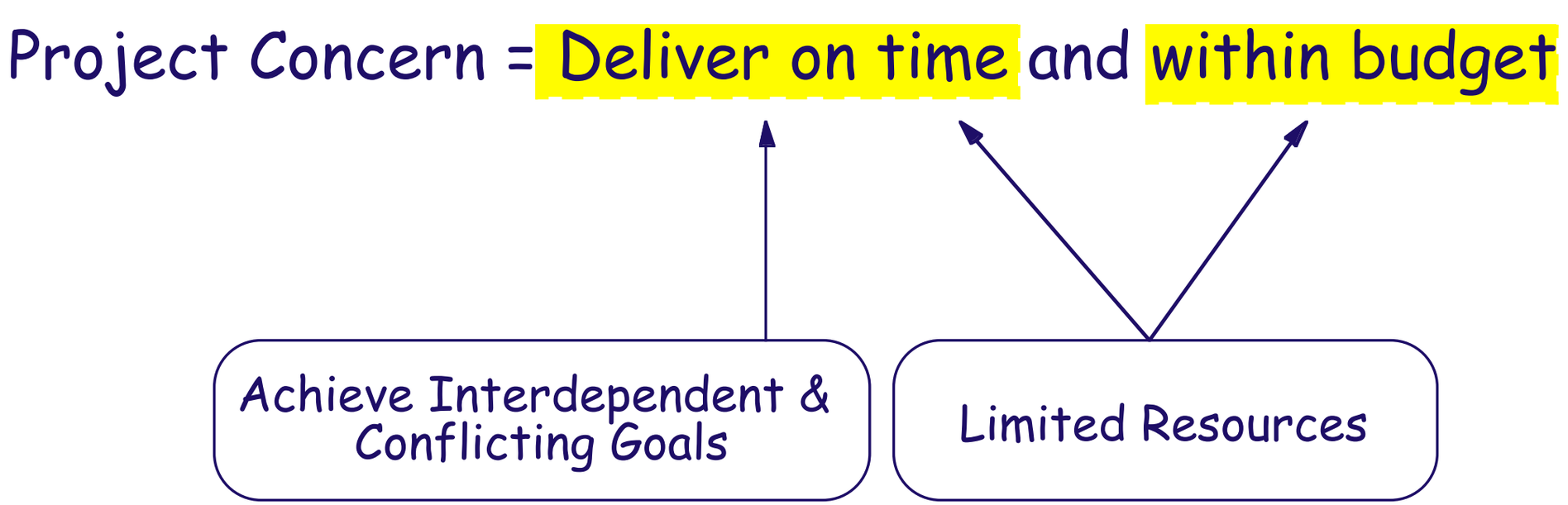
Roadmap

- Project Management: Why and what?
- Risk management
- Scoping and estimation, planning and scheduling
- Dealing with delays
- Staffing, directing, teamwork

Why and What?

Why Project Management?

- Almost all software products are obtained via *projects*. (as opposed to manufactured products)



The Project Team is the primary Resource!

What is Project Management?

Plan the work and work the plan

- Management Functions
  - Planning: Estimate and schedule resources
  - Organization: Who does what
  - Staffing: Recruiting and motivating personnel
  - Directing: Ensure team acts as a whole
  - Monitoring (Controlling): Detect plan deviations + corrective actions

Scope, estimation, planning and scheduling

Focus on Scope

> *For decades, programmers have been whining, "The customers can't tell us what they want. When we give them what they say they want, they don't like it." Get over it. This is an absolute truth of software development. The requirements are never clear at first. Customers can never tell you exactly what they want.*
>
> — Kent Beck

Myth: Scope and Objectives

Myth

"A general statement of objectives is enough to start coding."

Reality

Poor up-front definition is the major cause of project failure.

Scope and Objectives

- In order to plan, you must set clear scope & objectives
  - Objectives identify the general goals of the project, not how they will be achieved.
  - Scope identifies the primary functions that the software is to accomplish, and bounds these functions in a quantitative manner.
- Goals must be realistic and measurable
  - Constraints, performance, reliability must be explicit
  - Customer must set priorities

Cost Estimation

- activity concerned with the estimation of the necessary resources for the realisation of the project
- Fundamental questions:
  - How much effort is required to complete an activity?
  - How much calendar time is needed to complete an activity?
  - What is the total cost of an activity?
  - Project estimation and scheduling are interleaved management activities

Estimation Strategies

These strategies are simple but risky:

Expert judgement	Consult experts and compare estimates *cheap, but unreliable*
Estimation by analogy	Compare with other projects in the same application domain *limited applicability*
Parkinson's Law	Work expands to fill the time available *pessimistic management strategy*
Pricing to win	You do what you can with the budget available *requires trust between parties*

Estimation Techniques

"Decomposition" and "Algorithmic cost modeling" are used together

Decomposition	Estimate costs for components + integration ☞ *top-down or bottom-up estimation*
Algorithmic cost modeling	Exploit database of historical facts to map size on costs ☞ *requires correlation data*

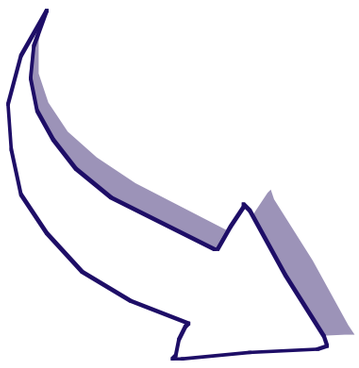
Top-down and bottom-up estimation

- Any of these approaches may be used top-down or bottom-up
- Top-down
  - Start at the system level and assess the overall system functionality and how this is delivered through sub-systems.
- Bottom-up
  - Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.

Measurement-based Estimation

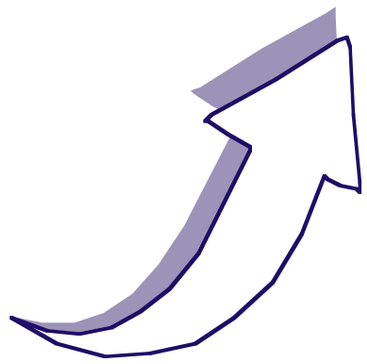
A. Measure

Develop a *system model* and measure its size



B. Estimate

Determine the effort with respect to an *empirical database* of measurements from *similar projects*



C. Interpret

Adapt the effort with respect to a specific *Development Project Plan*

Estimation and Commitment

Example: The XP process

1 a. Customers *write stories* and

b. Programmers *estimate stories*

else ask the customers to split/rewrite stories

2 Programmers *measure the team load factor*, the ratio of ideal programming time to the calendar

3 Customers *sort stories by priority*

4 Programmers *sort stories by risk*

5 a. Customers pick date, programmers calculate budget, customers pick stories adding up to that number, *or*

b. Customers pick stories, programmers calculate date

(customers complain, programmers ask to reduce scope, customers complain some more but reduce scope anyway,...)

Planning and Scheduling

- Good planning depends largely on project manager's intuition and experience!
- Split project into tasks.
  - Tasks into subtasks etc.
- For each task, estimate the time.
  - Define tasks small enough for reliable estimation.
- Significant tasks should end with a milestone.
  - Milestone = A verifiable goal that must be met after task completion
  - Clear unambiguous milestones are a necessity!

    ("80% coding finished" is a meaningless statement)
  - Monitor progress via milestones

Planning and Scheduling ...

- Define dependencies between project tasks
  - Total time depends on longest (= critical) path in activity graph
    - Minimize task dependencies to avoid delays
- Organize tasks concurrently to make optimal use of workforce
- Planning is iterative
  - monitor and revise schedules during the project!

Myth: Deliverables and Milestones

Myth

"The only deliverable for a successful project is the working program."

Reality

Documentation of all aspects of software development are needed to ensure maintainability.

Deliverables and Milestones

- Project deliverables are results that are delivered to the customer. For example:
  - initial requirements document
  - UI prototype
  - architecture specification
- Milestones and deliverables help to monitor progress
  - Should be scheduled roughly every 2-3 weeks
- NB: Deliverables must evolve as the project progresses!

Example: Task Durations and Dependencies

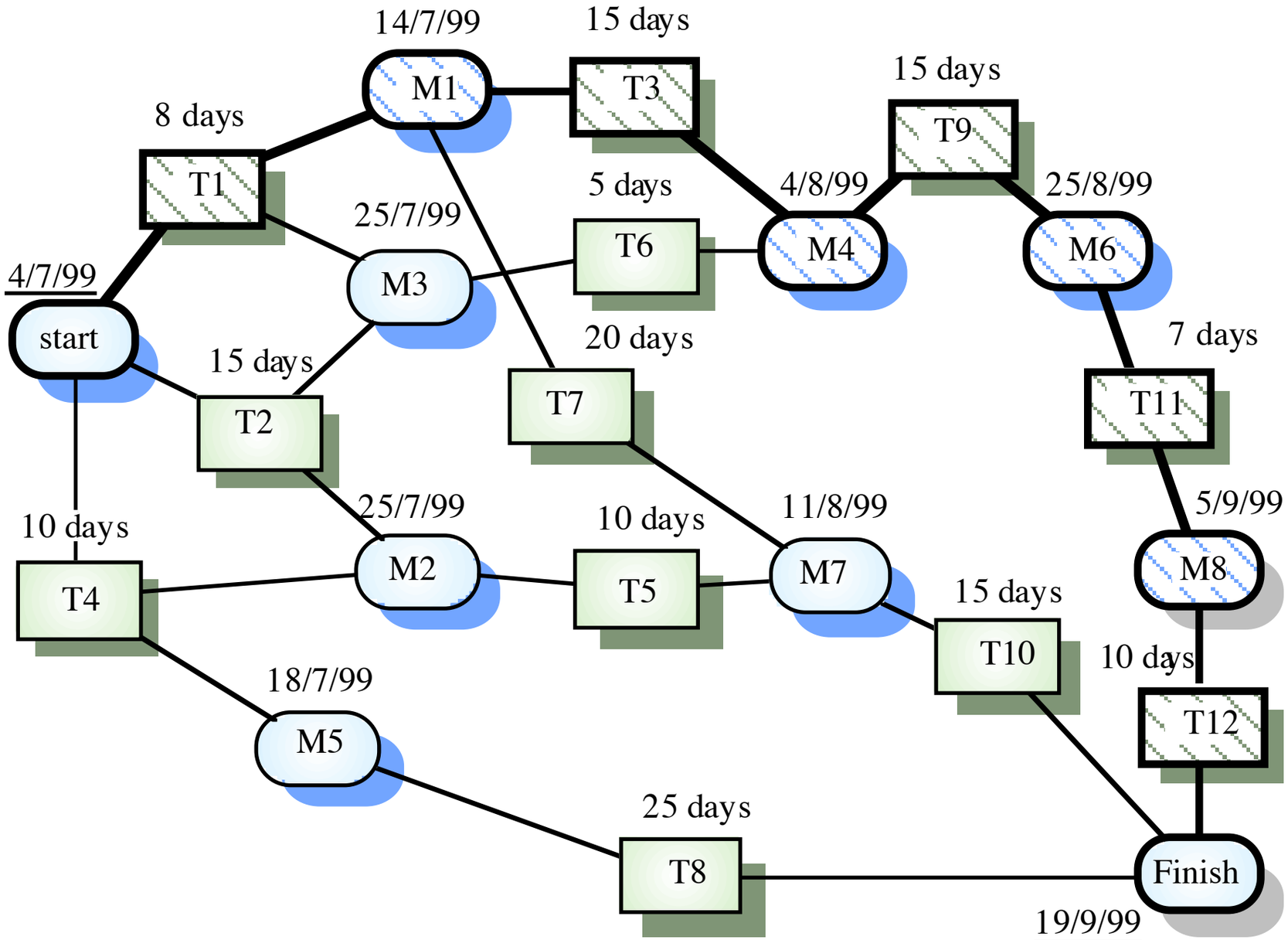
Task	*Duration (days)*	*Dependencies*
T1	8	
T2	15	
T3	15	T1
T4	10	
T5	10	T2, T4
T6	5	T1, T2
T7	20	T1
T8	25	T4
T9	15	T3, T6
T10	15	T5, T7
T11	7	T9
T12	10	T11

Example: Task Durations and Dependencies

What is the minimum total duration of this project?

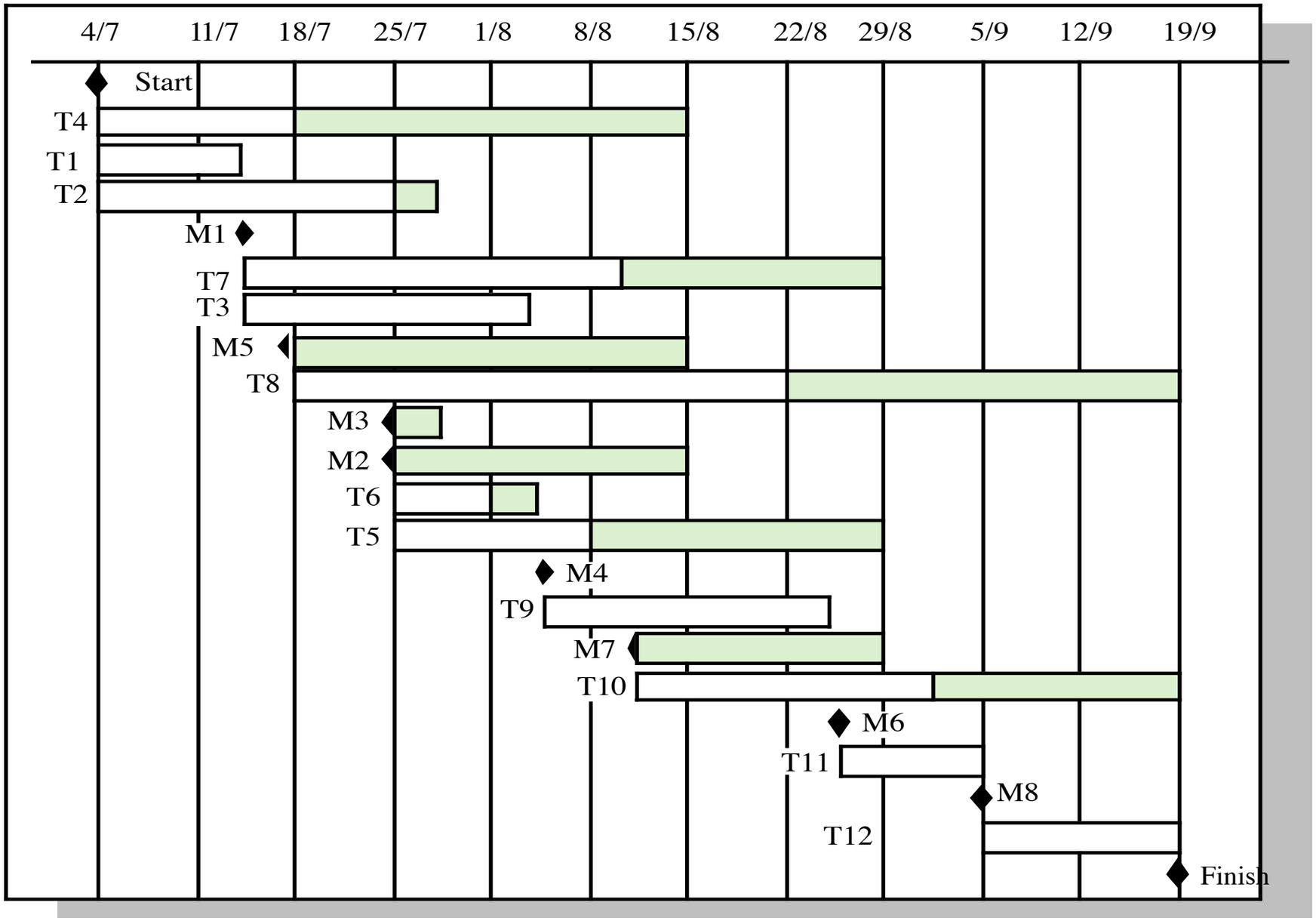
Task	***Duration (days)***	***Dependencies***
T1	8	
T2	15	
T3	15	T1
T4	10	
T5	10	T2, T4
T6	5	T1, T2
T7	20	T1
T8	25	T4
T9	15	T3, T6
T10	15	T5, T7
T11	7	T9
T12	10	T11

Pert Chart: Activity Network

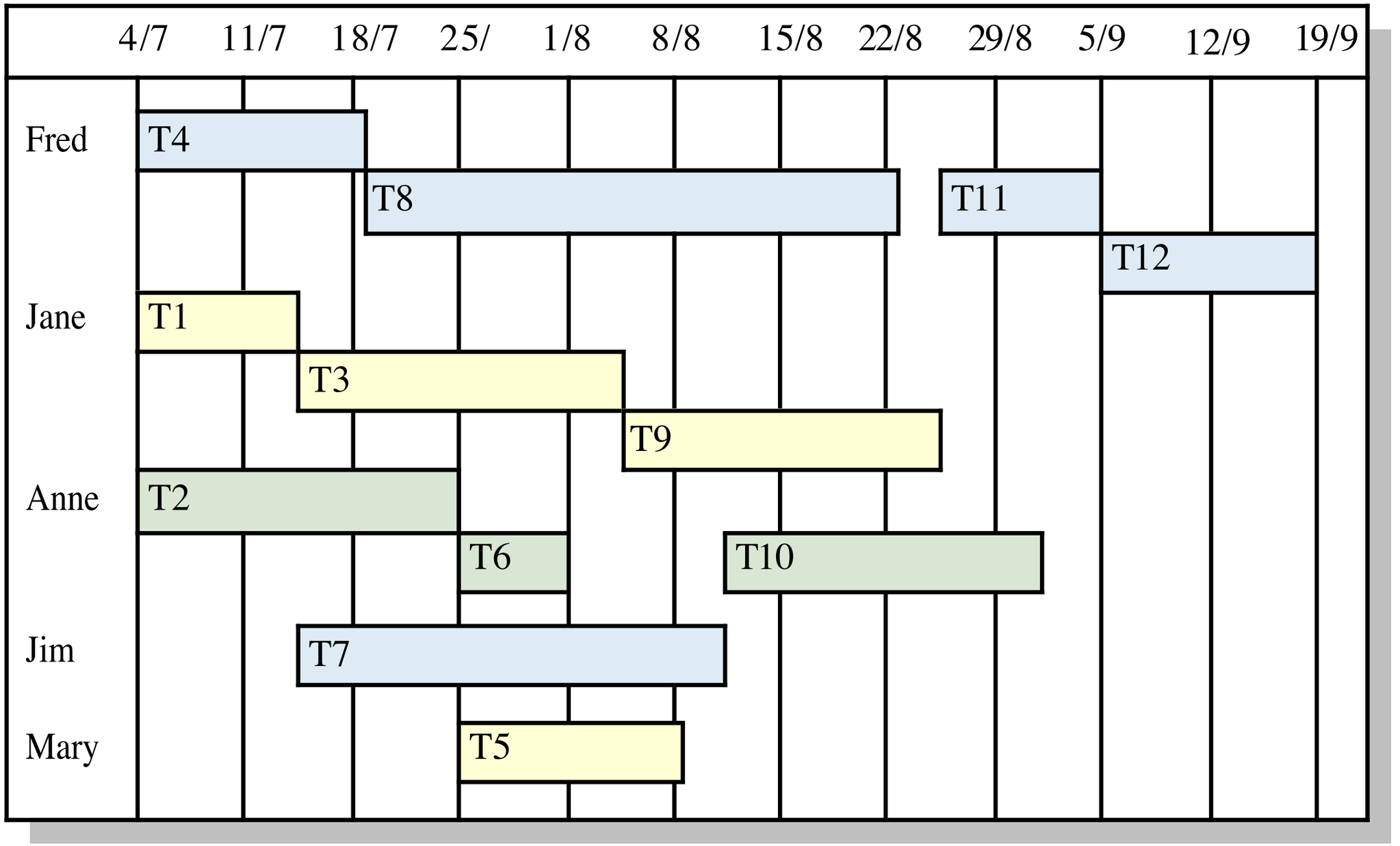


55 jours dans le chemin critique (8+15+15+7+10)

Gantt Chart: Activity Timeline



Gantt Chart: Staff Allocation



Delays

Myth: Delays

Myth

"If we get behind schedule, we can add more programmers and catch up."

Reality

Adding more people typically slows a project down.

Scheduling problems

- Estimating the difficulty of problems and the cost of developing a solution is hard.
- Productivity is not proportional to the number of people working on a task.
- Adding people to a late project makes it later due to communication overhead
- The unexpected always happens. Always allow contingency in planning.
- Cutting back in testing and reviewing is a recipe for disaster.
- Working overnight? Only short term benefits!

Planning under uncertainty

- State clearly what you know and don't know
- State clearly what you will do to eliminate unknowns
- Make sure that all early milestones can be met
- Plan to replan

Dealing with Delays

- Spot potential delays as soon as possible
  ... then you have more time to recover

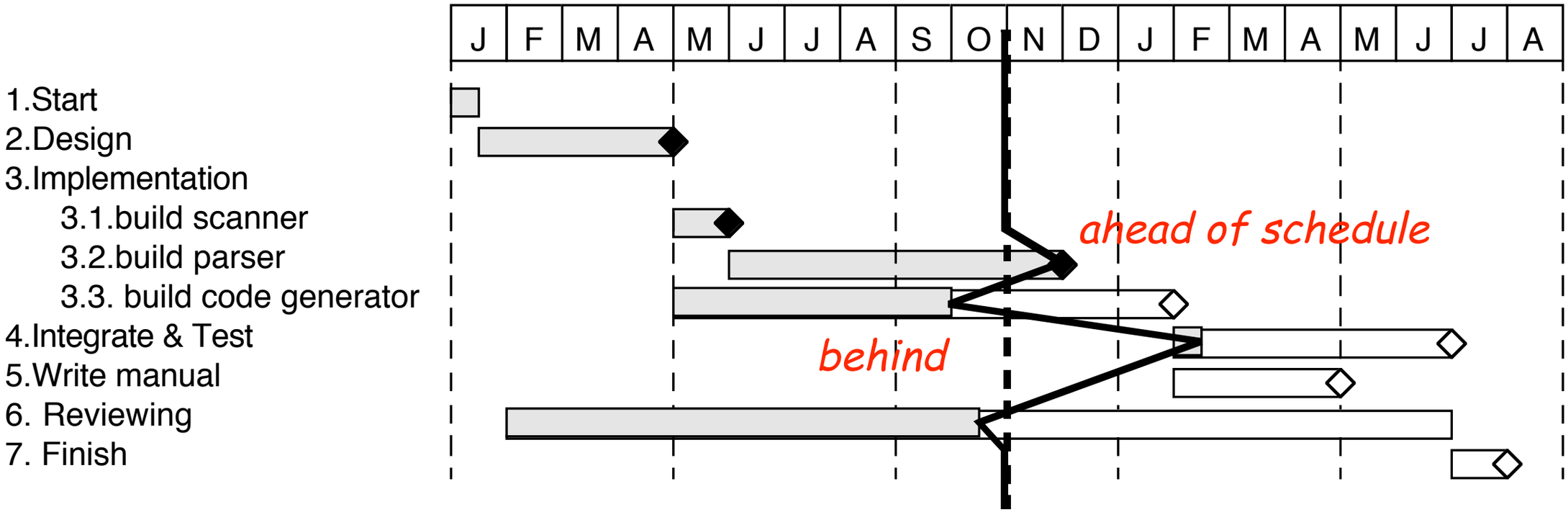
- How to spot?
  - Slippage analyse
    - use slip lines
  - Earned value analysis
    - planned time is the project budget
    - time of a completed task is credited to the project budget

Dealing with Delays ...

- How to recover?
- A combination of following 3 actions
  - Adding senior staff for well-specified tasks
    - outside critical path to avoid communication overhead
  - Prioritize requirements and deliver incrementally
    - deliver most important functionality on time
    - testing remains a priority (even if customer disagrees)
  - Extend the deadline

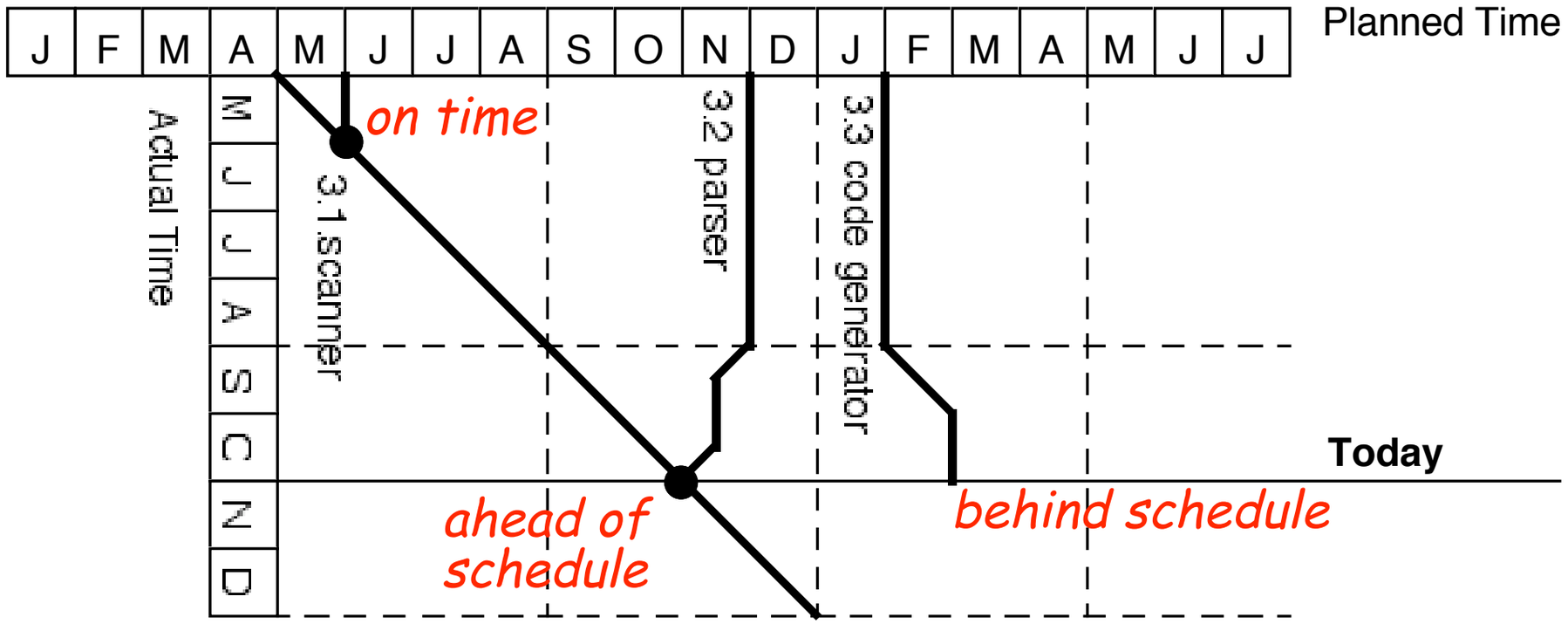
Gantt Chart: Slip Line visualizes slippage

- Shade time line = portion of task completed
- Draw a slip line at current date, connecting endpoints of the shaded areas
  - bending to the right = ahead of schedule
  - to the left = behind schedule



Timeline Chart: visualise slippage evolution

- downward lines represent planned completion time as they vary in current time
- bullets at the end of a line represent completed tasks



Slip Line vs. Timeline

Slip Line	Monitors *current slip status* of project tasks many tasks only for 1 point in time include a few slip lines from the past to illustrate evolution
Timeline	Monitors how the slip status of project tasks *evolves* few tasks crossing lines quickly clutter the figure colours can be used to show more tasks complete time scale

Risk Management

Example: Ariane 5

Destruction after 37 seconds flight on the
4th of June 1996 at Kourou

- Risque could have been predicted
- Malfunction of the software
- One of the most expensive computer bugs in history

Risk Management

If you don't actively attack risks, they will actively attack you. Tom Gilb

- Project risks
  - budget, schedule, resources, size, personnel, morale ...
- Technical risks
  - implementation technology, verification, maintenance ...
- Business risks
  - market, sales, management, commitment ...

Risk Management …

- Management must:
  - identify risks as early as possible
  - assess whether risks are acceptable
  - take appropriate action to mitigate and manage risks
    - e.g., training, prototyping, iteration, ...
  - monitor risks throughout the project

Risk Management Techniques

Risk Items	***Risk Management Techniques***
Personnel *shortfalls*	Staffing with top talent; *team building*; cross-training; pre-scheduling key people
Unrealistic schedules and budgets	Detailed multi-source cost & schedule estimation; *incremental development*; reuse; re-scoping
Developing the *wrong* software functions	User-surveys; *prototyping*; early users' manuals

Risk Management Techniques

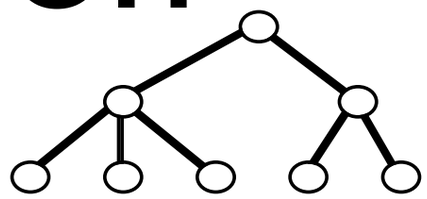
Risk Items	***Risk Management Techniques***
Continuing stream of *requirements changes*	High change threshold; information hiding; *incremental development*
Real time *performance* shortfalls	Simulation; benchmarking; modeling; prototyping; *instrumentation; tuning*
Straining computer science capabilities	Technical analysis; cost-benefit analysis; *prototyping*; reference checking

Staffing, directing, teamwork

Software Teams

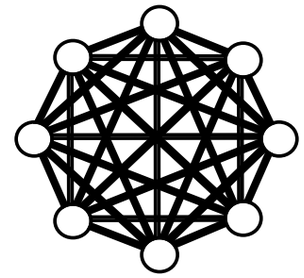
- Teams should be relatively small (< 8 members)
  - minimize communication overhead
  - team quality standard can be developed
  - members can work closely together
  - programs are regarded as team property ("egoless programming")
  - continuity can be maintained if members leave
- Break big projects down into multiple smaller projects (hierarchical decomposition)
- Small teams may be organised in an informal, democratic way (ego-less teams)
- Chief programmer teams try to make the most effective use of skills and experience

Hierarchical Decomposition



- Properties
  - each member reports to a manager and is responsible to carry out the tasks delegated by the manager
  - because every member is only responsible for his/her tasks, problems could remain unnoticed
  - best suited for big projects with a strict schema and where each person is competent and each role is well-defined

Ego-less Teams

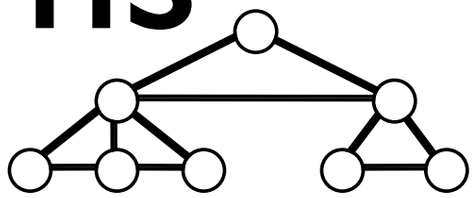


- Properties
  - group is more important than an individual
  - team works together to achieve a common goal
  - decisions are taken by consensus
  - every information is available for each member
  - suited for
    - difficult projects with lots of challenges
    - teams where each member is competent and has lots of experience.

Ego-less Teams

- Advantages
  - there is collaboration
  - quality standards of the group can be developed
  - each member knows the other members
  - each member knows what the other members are doing
  - each member can improve the programs of the other members.

Chief Programmer Teams



b) Programmeur-chef

- Consist of a kernel of specialists helped by others as required
  - chief programmer takes full responsibility for design, programming, testing and installation of system
  - backup programmer keeps track of CP's work and develops test cases
  - librarian manages all information
  - others may include: project administrator, toolsmith, documentation editor, language/system expert, tester, and support programmers …

Chief Programmer Teams

- Reportedly successful but problems are:
  - Can be difficult to find talented chief programmers
  - Might disrupt normal organisational structures
  - May be de-motivating for those who are not chief programmers

Directing Teams

Managers serve their team

Managers ensure that team has the *necessary information and resources*

"The manager's function is not to make people work, it is to make it possible for people to work"

— Tom DeMarco

Responsibility demands authority

Managers must *delegate*

Trust your own people and they will trust you.

Directing Teams

Managers manage

Managers cannot perform tasks on the *critical path*

Especially difficult for technical managers!

Developers control deadlines

A manager cannot meet a deadline to which the developers have not agreed

Team Roles

- Belbin introduced the notion of Team Roles
  - "A tendency to behave, contribute and interrelate with others in a particular way."
  - need people from different roles to be successful
    - Team of *only* experts will not work
    - Neither will team of only managers

What roles are proposed?

- Chair (co-ordinator and social leader)
- Shaper (gives drive and impetus)
- Plant/Innovator (ideas person)
- Monitor/evaluator (stopping over enthusiasm, missing key points)
- Resource investigator (delicate external negotiations)
- Organiser/company worker (implementer - turns ideas into practical action)
- Team worker (diffuses friction)
- Completer/Finisher (progress chaser)

References

- Ian Sommerville. Software Engineering, Addison-Wesley, Eighth Edn., 2007.
- R. Pressman. Software Engineering — A Practitioner's Approach, Mc-Graw Hill, Third Edn., 1994.
- F. Brooks. The Mythical Man-Month, Addison-Wesley, 1975
- T. Love. Object Lessons, SIGS Books, 1993
- A. Goldberg and K. Rubin. Succeeding with Objects: Decision Frameworks for Project Management, Addison-Wesley, 1995
- Kent Beck. Extreme Programming Explained: Embrace Change, Addison Wesley, 1999
- R.M. Belbin, Butterworth Heinemann. Management Teams - Why they succeed or fail, , reprint edition, 1996.