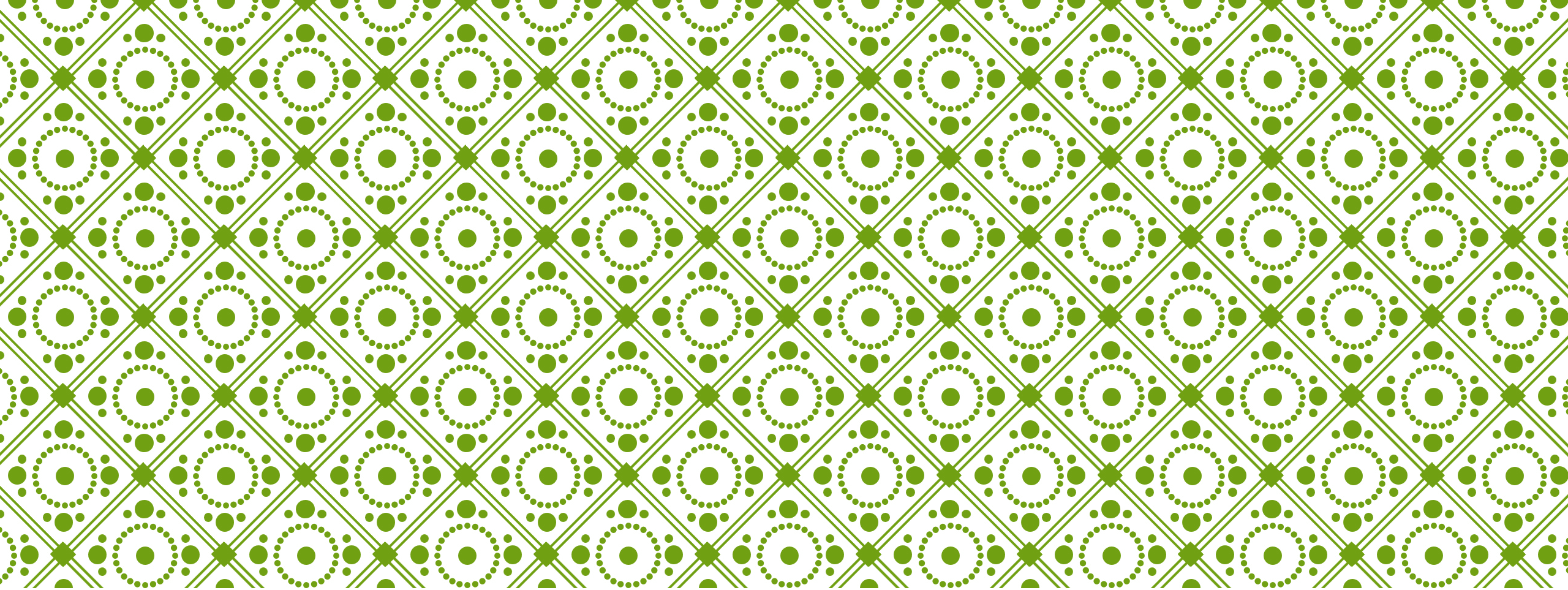
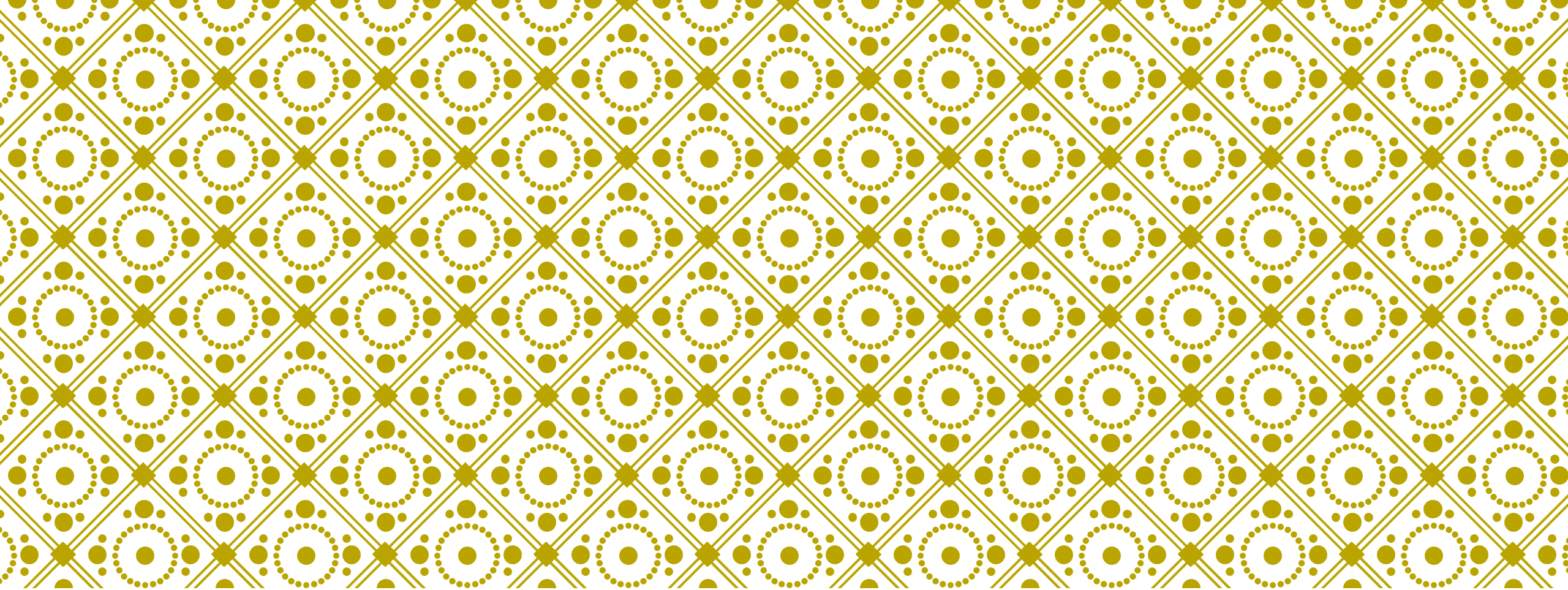


INTRODUCTION TO SOFTWARE AND SOFTWARE ENGINEERING

Lecture Notes prepared by
Asst. Prof. Melody Angelique C. Rivera
Faculty, College of Computer Studies,
Silliman University



FAQ ABOUT **SOFTWARE**

WHAT IS SOFTWARE?

Instructions (computer programs) that when executed, provide desired features, function, and performance

Data structures that enable the programs to adequately manipulate information

Descriptive information in both hard copy and virtual forms that describes the operation and use of the programs

NATURE OF SOFTWARE (1)

Software is both a ***product*** and a ***vehicle*** that delivers a product

As a **product**, software is an information transformer

- producing, managing, acquiring, modifying, displaying, or transmitting information

As the **vehicle**, software is used to deliver the product

- software acts as the basis for
  - the control of the computer (operating systems)
  - the communication of information (networks), and
  - the creation and control of other programs (software tools and environments)

NATURE OF SOFTWARE (2)

Software delivers the most important product of our time—***information***

It transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context;

It manages business information to enhance competitiveness;

It provides a gateway to worldwide information networks (e.g., the Internet), and provides the means for acquiring information in all of its forms

SOFTWARE CHARACTERISTICS

Software is **developed** or **engineered**

- it is not manufactured in the classical sense

Software **doesn't "wear out."** (but it deteriorates)

Although the industry is moving toward component-based construction, most software continues to be custom-built

WHAT ARE THE ATTRIBUTES OF GOOD SOFTWARE? (1)

Good software should

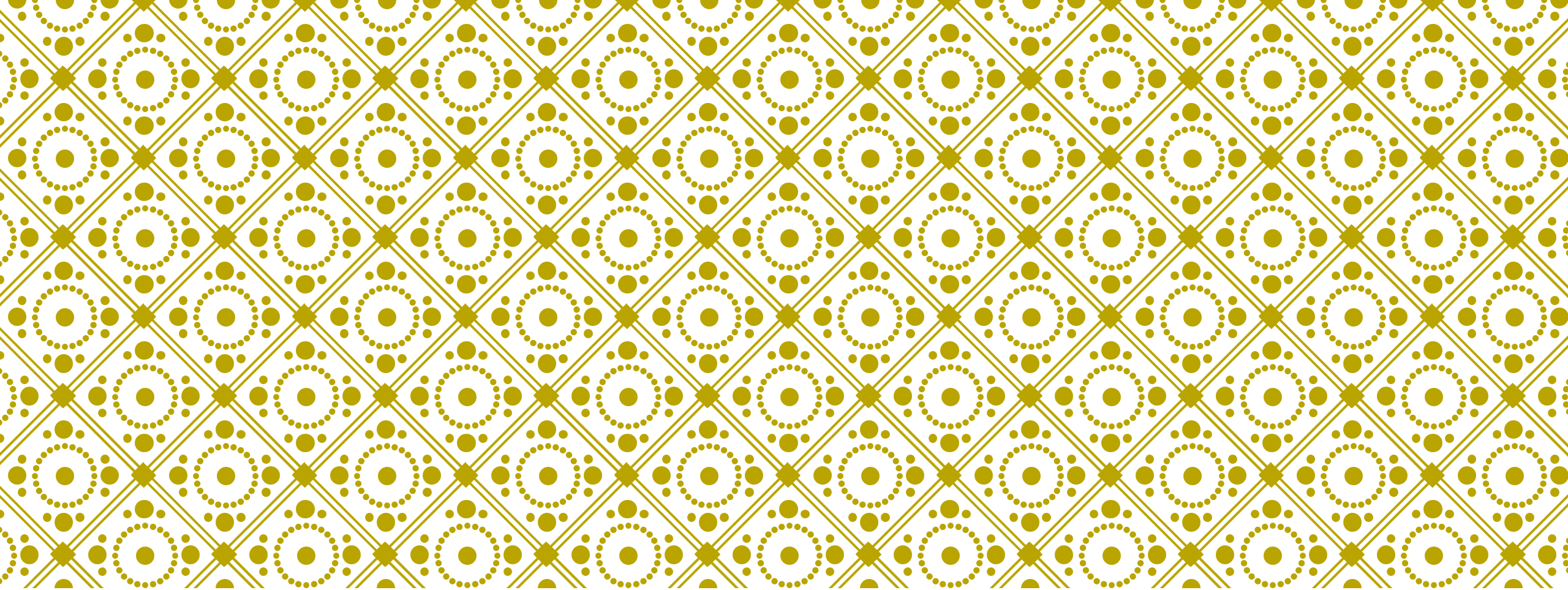
- deliver the required functionality and performance to the user
- be maintainable, dependable, and usable

Essential attributes:

- Maintainability
- Dependability and security
- Efficiency
- Acceptability

WHAT ARE THE ATTRIBUTES OF GOOD SOFTWARE? (2)

Product characteristics	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.



FAQ ABOUT **SOFTWARE ENGINEERING**

WHAT IS SOFTWARE ENGINEERING?

An engineering discipline that is concerned with **all aspects of software production** from the early stages of system specification to maintaining the system after it has gone into use

WHAT ARE THE FUNDAMENTAL SOFTWARE ENGINEERING ACTIVITIES?

software specification

- where customers and engineers define the software that is to be produced and the constraints on its operation

software development

- where the software is designed and programmed

software validation

- where the software is checked to ensure that it is what the customer requires

software evolution

- where the software is modified to reflect changing customer and market requirements

WHAT IS THE DIFFERENCE BETWEEN SOFTWARE ENGINEERING AND COMPUTER SCIENCE?

Computer science focuses on theory and fundamentals

Software engineering is concerned with the practicalities of developing and delivering useful software

WHAT IS THE DIFFERENCE BETWEEN SOFTWARE ENGINEERING AND SYSTEM ENGINEERING?

System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering

Software engineering is **part** of this more general process

WHAT ARE THE KEY CHALLENGES FACING SOFTWARE ENGINEERING?

Coping with increasing diversity

Coping with demands for reduced delivery times

Developing trustworthy software

WHAT ARE THE COSTS OF SOFTWARE ENGINEERING?

Approximately:

60% of software costs are **development costs**

40% are **testing costs**

For custom software, **evolution costs** often exceed development costs

WHAT ARE THE BEST SOFTWARE ENGINEERING TECHNIQUES AND METHODS?

While all software projects have to be professionally managed and developed, **different techniques are appropriate for different types of systems**

For example:

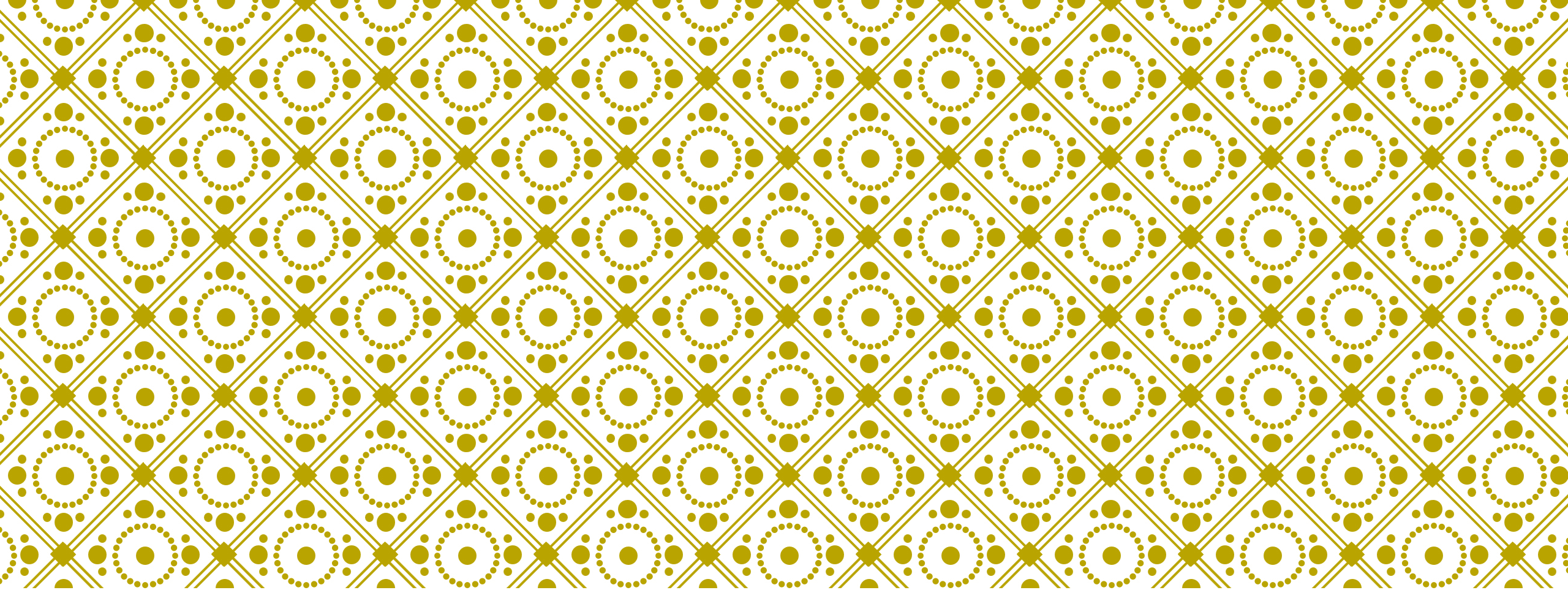
- games should always be developed using a series of prototypes
- safety critical control systems require a complete and analyzable specification to be developed

You cannot say that one method is better than another

WHAT DIFFERENCES HAS THE WEB MADE TO SOFTWARE ENGINEERING?

The Web has led to the **availability of software services** and the possibility of **developing highly distributed service-based systems**

Web-based systems development has led to important advances in programming languages and software reuse



SOFTWARE APPLICATION DOMAINS (TYPES OF APPLICATIONS)

SYSTEM SOFTWARE (1)

a collection of programs written to service other programs

some system software processes complex (but determinate) information structures

- compilers, editors, and file management utilities

Software is ***determinate*** if the order and timing of inputs, processing, and outputs is **predictable**

Software is ***indeterminate*** if the order and timing of inputs, processing, and outputs **cannot be predicted in advance**

SYSTEM SOFTWARE (2)

other systems applications process largely indeterminate data

- operating system components, drivers, networking software, telecommunications processors

characterized by

- heavy interaction with computer hardware;
- heavy usage by multiple users;
- concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures;
- multiple external interfaces

APPLICATION SOFTWARE

stand-alone programs that solve a specific business need

process business or technical data that facilitates business operations or management/technical decision making

is used to control business functions in real time (e.g., point-of-sale transaction processing, real-time manufacturing process control)

ENGINEERING/SCIENTIFIC SOFTWARE

Characterized by “number crunching” algorithms

It ranges from

- astronomy to volcanology
- automotive stress analysis to space shuttle orbital dynamics
- from molecular biology to automated manufacturing

Modern applications are moving away from conventional numerical algorithms

- Computer-aided design (CAD), system simulation, and other interactive applications have begun to take on real-time and system software characteristics

EMBEDDED SOFTWARE

resides within a product or system

is used to implement and control features and functions for the end user and for the system itself

can perform limited and esoteric functions

- e.g., key pad control for a microwave oven

provide significant function and control capability

- e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems

PRODUCT-LINE SOFTWARE

designed to provide a specific capability for use by many different customers

can focus on a limited and esoteric marketplace

- e.g., inventory control products

address mass consumer markets

- e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications

WEB APPLICATIONS (1)

Also called “WebApps”

A network-centric software category that spans a wide array of applications

Simplest form – a set of linked hypertext files that present information using text and limited graphics

With the emergence of Web 2.0, WebApps evolved into sophisticated computing environments

- provide stand-alone features, computing functions, and content to the end user
- integrated with corporate databases and business applications

WEB APPLICATIONS (2)

Web 2.0

(from Wikipedia)

- also known as Participative (or Participatory) and Social Web
- refers to websites that emphasize user-generated content, ease of use, participatory culture and interoperability (i.e., compatible with other products, systems, and devices) for end users

(from Britannica)

- term for the Web dominated by social networking, user-generated content, and cloud computing

Examples:

- Wikipedia, Facebook, Youtube

WEB APPLICATIONS (3)

Web 3.0

(from W3C)

- The term “Semantic Web” refers to W3C’s vision of the Web of linked data or “Web of data,” the sort of data you find in databases
- The ultimate goal is to enable computers to do more useful work and to develop systems that can support trusted interactions over the network
- Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data
- Linked data are empowered by technologies such as RDF, SPARQL, OWL, and SKOS

WEB APPLICATIONS (4)

Web 4.0

(from IGI Global)

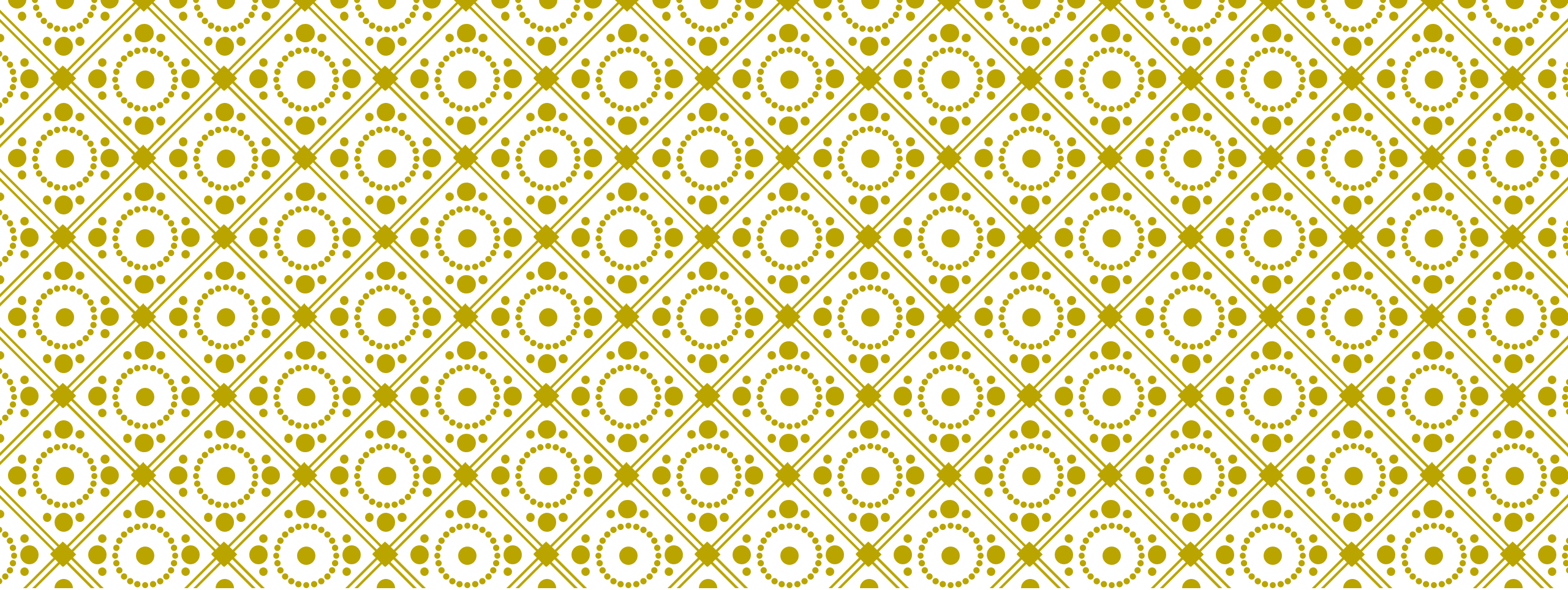
- the forth phase in Web's evolution; also known as "intelligent Web" or "smart Web"
- services will be autonomous, proactive, content-exploring, self-learning, collaborative, and content-generating agents based on fully matured semantic and reasoning technologies as well as AI
- software agent(s) roaming on the Internet or residing on your computer could reason and communicate with other such agents and systems and work collaboratively to accomplish things on your behalf
- In 2009, Tim O'Reilly and John Battelle, coined the term "Web Squared" or Web meets World (the notion of using the Web to address real-world problems)
  - The idea is that to solve the world's most pressing problems, the power of the Web will be put to work (technologies, business models, philosophies of openness, collective intelligence, and transparency

ARTIFICIAL INTELLIGENCE SOFTWARE

makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis

Include

- Robotics
- expert systems
- pattern recognition (image and voice)
- artificial neural networks
- theorem proving
- game playing



ATTRIBUTES IN MOST WEBAPPS

NETWORK INTENSIVENESS

A WebApp resides on a network and must serve the needs of a diverse community of clients

The network may enable worldwide access and communication (i.e., the Internet) or more limited access and communication (e.g., a corporate Intranet)

CONCURRENCY

A large number of users may access the WebApp at one time

In many cases, the patterns of usage among end users will vary greatly

UNPREDICTABLE LOAD

The number of users of the WebApp may vary by orders of magnitude from day to day

- For example, 100 users may show up on Monday, while 10,000 may use the system on Thursday

PERFORMANCE

If a WebApp user must wait too long, he or she may decide to go elsewhere

AVAILABILITY

Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a 24/7/365 basis

- Users in Australia or Asia might demand access during times when traditional domestic software applications in North America might be taken off-line for maintenance

DATA-DRIVEN

The primary function of many WebApps is to use **hypermedia** to present text, graphics, audio, and video content to the end user

WebApps are commonly used to access information that exists on databases that are not an integral part of the Web-based environment (e.g., e-commerce or financial applications)

CONTENT-SENSITIVE

The **quality and aesthetic nature of content** remains an important determinant of the quality of a WebApp

CONTINUOUS EVOLUTION

Web applications evolve continuously

- Unlike conventional application software that evolves over a series of planned, chronologically spaced releases

It is not unusual for the content of some WebApps to be updated on a minute-by-minute schedule or for content to be independently computed for each request

IMMEDIACY

The compelling need to get software to market quickly

Also a characteristic of many application domains

WebApps often exhibit a time-to-market that can be a matter of a few days or weeks

SECURITY

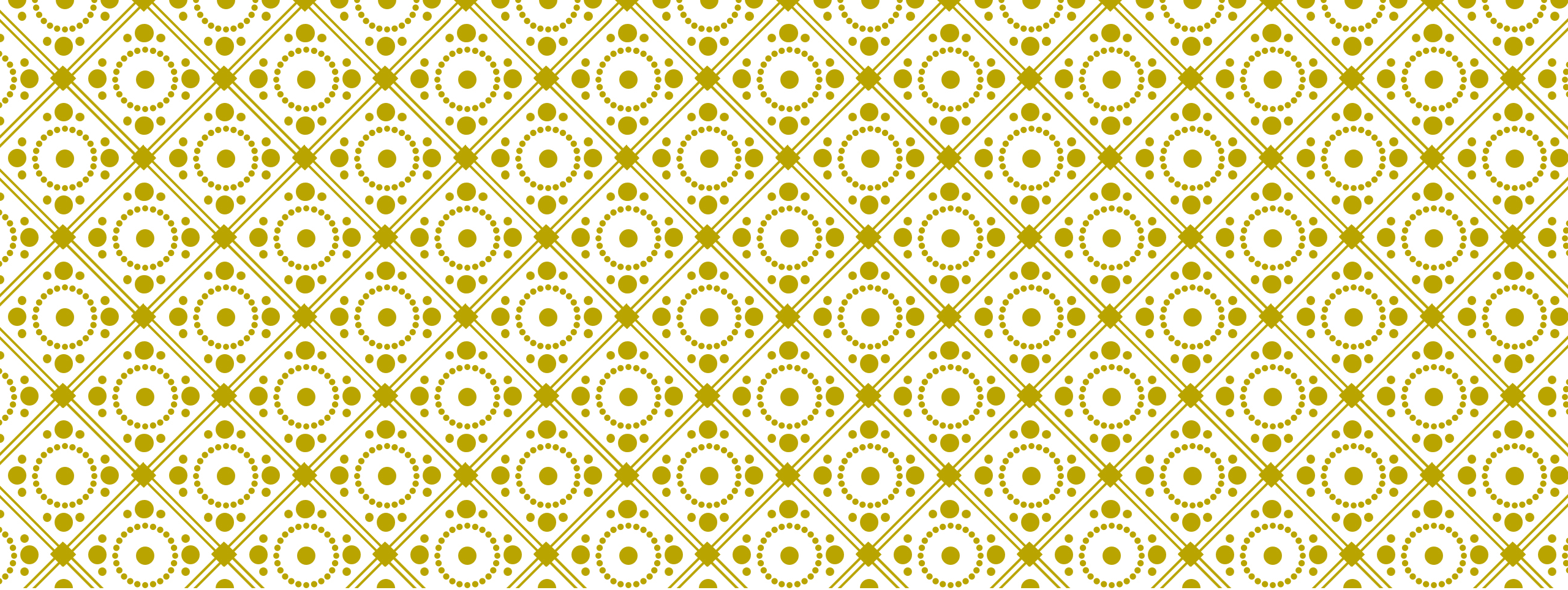
Because WebApps are available via network access, it is difficult, to limit the population of end users who may access the application

To **protect sensitive content** and **provide secure modes of data transmission,** strong security measures must be implemented throughout the infrastructure that supports a WebApp and within the application itself

AESTHETICS

A WebApp's look and feel

When an application has been designed to market or sell products or ideas, aesthetics may have as much to do with success as technical design



SOFTWARE ENGINEERING PRINCIPLES

THE FIRST PRINCIPLE: **THE REASON IT ALL EXISTS**

A software system exists for one reason: ***to provide value to its users***

- All decisions should be made with this in mind

Before starting anything, ask yourself questions such as: “Does this add real value to the system?”

If the answer is “no,” don’t do it.

All other principles support this principle

THE SECOND PRINCIPLE: **KISS** (KEEP IT SIMPLE, STUPID!)

Software design is not a haphazard process

There are many factors to consider in any design effort

All design should be as simple as possible, but no simpler.

- this facilitates having a more easily understood and easily maintained system

The more elegant designs are usually the more simple one

It often takes a lot of thought and work over multiple iterations to simplify

- the payoff is software that is more maintainable and less error-prone

THE THIRD PRINCIPLE: **MAINTAIN THE VISION**

A clear vision is essential to the success of a software project.

- Without one, a project almost unfailingly ends up being "of two [or more] minds" about itself
- Without conceptual integrity, a system threatens to become a patchwork of incompatible designs, held together by the wrong kind of screws. . . .

The architectural vision of a software system must not be compromised

Having an empowered architect who can hold the vision and enforce compliance helps ensure a very successful software project

THE FOURTH PRINCIPLE: **WHAT YOU PRODUCE, OTHERS WILL CONSUME**

In some way or other, someone else will use, maintain, document, or depend on being able to understand your system

Always specify, design, and implement knowing someone else will have to understand what you are doing

Specify (with an eye to the users)

Design (keeping the implementers in mind)

Code (with concern for those who must maintain and extend the system)

- Someone may have to debug the code you write, and that makes them a user of your code
- Making their job easier adds value to the system

THE FIFTH PRINCIPLE: **BE OPEN TO THE FUTURE**

A system with a long lifetime has more value

In today's computing environments, **software lifetimes** are **typically measured in months** instead of years

- specifications change on a moment's notice
- hardware platforms are obsolete just a few months old
- True "industrial-strength" software systems must endure far longer

Never design yourself into a corner.

- Always ask "what if," and prepare for all possible answers by creating systems that solve the general problem, not just the specific one

THE SIXTH PRINCIPLE: **PLAN AHEAD FOR REUSE**

Reuse saves time and effort but requires forethought and planning

Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated

Achieving a high level of reuse is the hardest goal to accomplish in developing a software system

This is a major benefit of using object-oriented technologies but the return on investment is not automatic

THE SEVENTH PRINCIPLE: **THINK!**

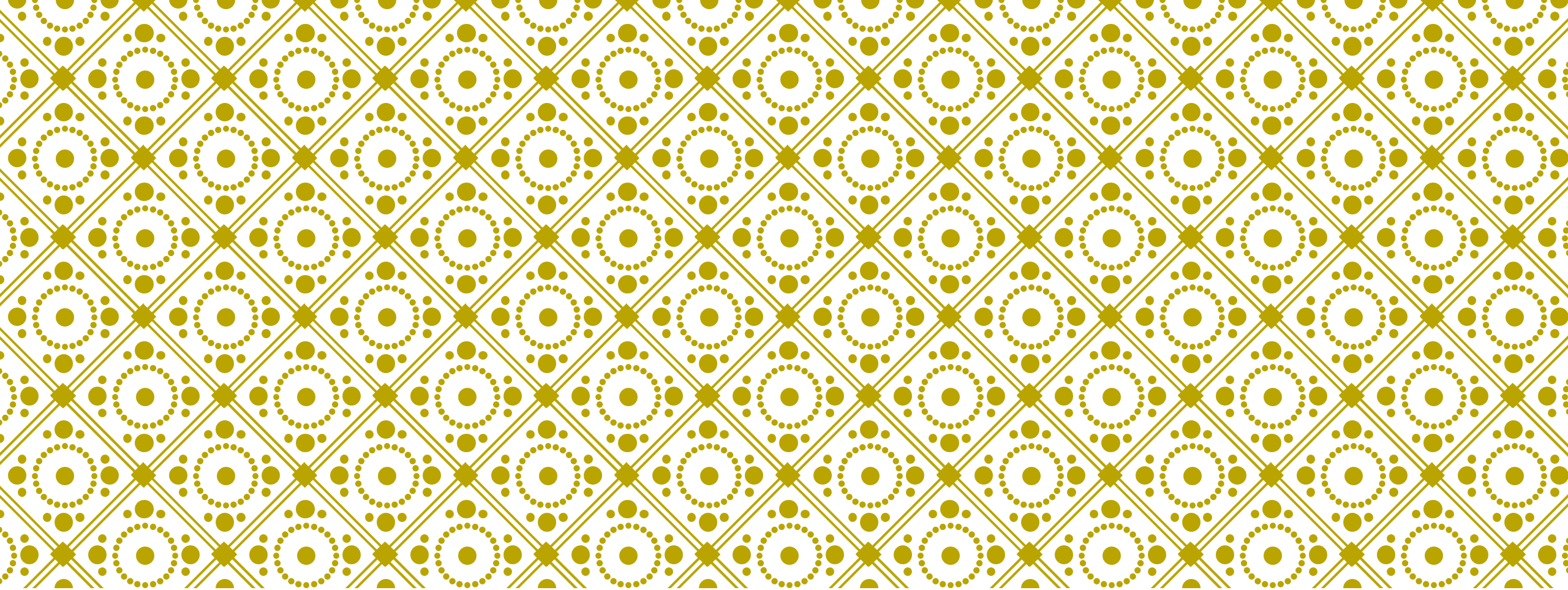
This last principle is the most overlooked

When you think about something, you are more likely to do it right and gain knowledge about how to do it right again

If you do think about something and still do it wrong, it becomes a valuable experience

A side effect of thinking is learning to recognize when you don't know something, at which point you can research the answer

When clear thought has gone into a system, value comes out



SOFTWARE ENGINEERING ETHICS

CONFIDENTIALITY

You should normally respect the confidentiality of your employers or clients irrespective of whether or not a formal confidentiality agreement has been signed

COMPETENCE

You should not misrepresent your level of competence

You should not knowingly accept work that is outside your competence

INTELLECTUAL PROPERTY RIGHTS

You should be **aware of local laws** governing the use of **intellectual property** such as **patents** and **copyright**

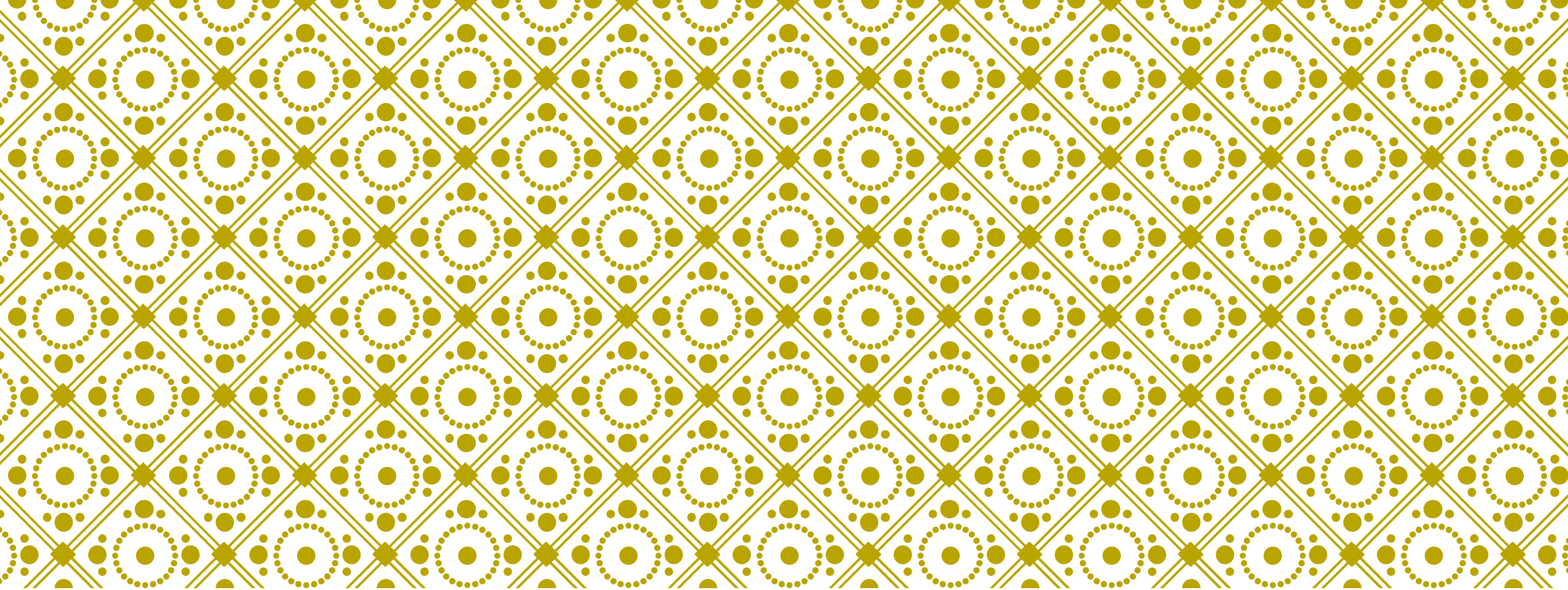
You should be careful to ensure that the intellectual property of employers and clients is protected

COMPUTER MISUSE

You should not use your technical skills to misuse other people's computers

Computer misuse ranges

- from relatively trivial (game playing on an employer's machine)
- to extremely serious (dissemination of viruses or other malware)



END OF PRESENTATION…